I.J. Information Engineering and Electronic Business, **2025, 2, 1-50**
Published Online on April 8, 2025 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2025.02.01

Agile Methodology of Information Engineering for Semantic Annotations Categorization and Creation in Scientific Articles Based on NLP and Machine Learning Methods

Danylo Levkivskyi
Department of Information Systems and Networks, Institute of Computer Sciences and Information Technologies, Lviv Polytechnic National University, Lviv, 79013, Ukraine
E-mail: danyil.levkivskyi.sa.2020@lpnu.ua
ORCID iD: https://orcid.org/0009-0004-8828-303X

Victoria Vysotska
Department of Information Systems and Networks, Institute of Computer Sciences and Information Technologies, Lviv Polytechnic National University, Lviv, 79013, Ukraine
E-mail: Victoria.A.Vysotska@lpnu.ua
ORCID iD: https://orcid.org/0000-0001-6417-3689

Lyubomyr Chyrun
Applied Mathematics Department, Faculty of Applied Mathematics and Informatics, Ivan Franko National University of Lviv, Lviv, 79000, Ukraine
E-mail: Lyubomyr.Chyrun@lnu.edu.ua
ORCID iD: https://orcid.org/0000-0002-9448-1751

Yuriy Ushenko*
Department of Physics, Shaoxing University, Shaoxing, Zhejiang Province 312000, China
Department of Computer Science, Educational and Research Institute of Physical, Technical and Computer Sciences, Yuriy Fedkovych Chernivtsi National University, 58012, Ukraine
E-mail: y.ushenko@chnu.edu.ua
ORCID iD: https://orcid.org/0000-0003-1767-1882
*Corresponding Author

Dmytro Uhryn
Department of Computer Science, Educational and Research Institute of Physical, Technical and Computer Sciences, Yuriy Fedkovych Chernivtsi National University, 58012, Ukraine
E-mail: d.ugryn@chnu.edu.ua
ORCID iD: https://orcid.org/ 0000-0003-4858-4511

Cennuo Hu
Department of Computer Science, College of Science, Purdue University, West Lafayette, IN 47907, USA
E-mail: hu945@purdue.edu
ORCID iD: https://orcid.org/0009-0008-9589-9253

Received: 19 January, 2025; Revised: 25 February, 2025; Accepted: 20 March, 2025; Published: 08 April, 2025

Abstract: Research devoted to the categorization and creation of semantic annotations for scientific articles stands out as an essential direction of development in the context of the growing volume of scientific literature. The application of machine learning and natural language processing in this field allows you to effectively organize and provide access to scientific information. The article discusses methods of automatic annotation of texts. Based on the review, the use of the constraint propagation model is proposed to improve the technique of text relationship maps. The developed software system is aimed at automating the process of analysis and categorization of scientific materials, which opens the way to improving the speed and accuracy of searching for the necessary information for researchers. The use of

advanced machine learning models, such as roBERTa and RAG, ensures the highest quality of data processing and creation of semantic annotations. The accuracy of predicting article categories after improving the model reached 88%**.** The novelty of the approach is the combination of categorization and semantic annotation to increase the convenience and speed of searching for scientific information. The software system opens up opportunities for future expansion and improvement through the use of advanced technologies and machine learning models. This study is noted for its relevance, originality of approach and potential for practical application in the field of scientific research and development of science as a whole. The proposed approach contributes to the development of the Information Engineering and Electronic Business industry through the following key aspects: automation of categorization and annotation of scientific articles, improving the accuracy of information search, increasing the efficiency of scientific research, and the flexibility and scalability of the solution.

Index Terms: Automatic Annotation, Constraint Propagation Model, Text Relationship Maps, TRM Method

1. Introduction

Categorization and creation of semantic annotations for scientific articles are becoming more and more relevant in the context of the constant growth of volumes of scientific literature [1-3]. Here are some key points that support the relevance of this topic [4-6]:

- *The volume of scientific information is increasing with the development of science and technology, and* the number of scientific publications is constantly increasing. It creates a need for tools that help efficiently deal with the growing volume of data.
- *Search for effective methods of information organization*. Categorization and annotation help organize information for quick and convenient access to it. It is essential for researchers looking for specific information from large volumes of data.
- *Increasing ease of search and analysis* as quick access to critical scientific articles can speed up research and development processes. Semantic annotations can help in a more accurate and faster search for relevant information.
- *Improving the quality of search*. The use of semantic technologies allows you to avoid problems related to the ambiguity of words or terms, which helps in improving the quality of search and analysis.

Therefore, in the context of the growing amount of scientific information and the constant need for convenient tools for searching and organizing this information, categorization and creation of semantic annotations for scientific articles are highly relevant and important areas of research.

The goal of our research is to develop a software system for the automatic creation of semantic annotations and categorization of scientific articles by using available ML technologies. The following tasks have been defined to achieve this goal:

- The task is to create software capable of automatically analysing the genders assigned to it in order to create semantic annotations and categorization. It means that the program will output a specific semantic annotation and category for the given article in text form. For this, the program needs built-in algorithms and methods that can help understand the context and obtain the necessary information from it. Also, the program needs built-in algorithms for data processing and cleaning, tools for working with LLM and, in general, a toolkit for conducting NLP operations.
- Extensive testing is required to ensure the reliability and efficiency of the program, which includes checking the validity of the data, checking the obtained results, and comparing the ready-made categories with the initial ones.
- Ensuring continuous improvement in the efficiency and accuracy of artificially generated text detection through constant improvement of the program.

The object of research in the context of "categorization and creation of semantic annotations for scientific articles" is scientific articles and their information environment. Let's look at a few key aspects of the research object:

- *Scientific articles* are the main object of analysis. Scientific articles are the primary means of transmitting scientific information and research results. They can be published in scientific journals, conference materials, collections of reports and other scientific publications.
- *Information environment*. The object of research is also the environment in which these scientific articles exist. It includes databases of scientific journals, websites of scientific publications, conference archives, bibliographic services and other sources where scientific information is stored and distributed.

- *Structure and content of articles*. The object of research is also the very structure and content of scientific articles. It includes the various components of the articles, such as the introduction, methodology, results, conclusions, and key terms, concepts, and other elements contained therein.
- *Ways of accessing scientific information*. The object of research can also be various ways of accessing scientific articles, including search engines, databases, and archives, as well as methods of reviewing and evaluating the quality of scientific publications.

The study of the research object will allow us to better understand the process of publication and dissemination of scientific information, as well as to develop more effective methods of categorization and annotation to improve the accessibility and search of this information. The subject of the research is the process of categorization of scientific articles and the development of semantic annotations for them. Categorization involves classifying articles according to various criteria, such as subject matter, research methodology, scope, etc. Semantic annotations consist of creating short descriptions that reflect the key aspects of articles and help in their search and understanding.

The process of categorization of scientific articles involves the systematization of these materials according to various criteria, which allows the creation of a structured database of scientific information. Categories can be created based on research topics, disciplines, research methods, level of complexity, etc. For example, articles can be classified by categories such as medicine, technology, and humanities or by specific thematic areas within each field.

Creating semantic annotations for scientific articles includes the extraction of key terms and the identification of central concepts and essential conclusions in each article. These annotations can be short textual descriptions that provide users with information about the content of articles without having to read them in full. It helps researchers find the information they need more quickly and determine whether an article meets their research needs.

Our program is a new combination of a vision of the necessary things to maintain a full-fledged base of scientific articles. For the first time, we will use categorization together in one place together with semantic annotations for a better understanding of the context and cataloguing of scientific articles. We will also use advanced technologies and models for machine learning, together with ready-made LLM models, which will allow us to expand the functionality of the program further for our purposes. This program uses a comprehensive approach to training models, as well as two models at once for different purposes, which will allow you to improve key aspects of its work separately. In addition, the powerful roBERTa model and the RAG method will be applied to the selected "OpenSource" LLM model, which will allow results to be obtained on the basis of chosen embeddings for specific articles. The main advantage of our program is its uniqueness, as well as the possibility of expansion. The used models improve their performance over time thanks to the work of their developers and allow you to migrate to other models over time without much effort if necessary.

The proposed approach contributes to the development of the Information Engineering and Electronic Business industry through the following key aspects:

1. *Automating the categorization and annotation of scientific articles*. The use of machine learning methods (in particular, the roBERTa and RAG models) allows you to automate the process of determining the categories of scientific articles and creating semantic annotations. It significantly simplifies access to relevant scientific information and speeds up the analysis of large volumes of text data.
2. *Improving the accuracy of information search*. The use of advanced natural language processing (NLP) algorithms and the constraint propagation model helps reduce the ambiguity of terms and improves the quality of scientific information search. It is essential for scientists who need quick access to narrowly focused materials.
3. *Increasing the efficiency of scientific research*. Automated processes of analysis and classification of articles reduce the workload on researchers, allowing them to find relevant publications faster. It contributes to more efficient use of scientific resources and simplifies the process of forming a literature review.
4. *Flexibility and scalability of the solution*. The developed system has a modular architecture that allows easy integration of new methods of text analysis and adaptation to changes in the formats of scientific publications. It makes it promising for further development and expansion of functionality.

Thus, the developed approach makes a significant contribution to the development of Information Engineering and Electronic Business, increasing the accessibility, accuracy and efficiency of working with scientific data.

2. Related works

2.1. Analytical review of developments and research in the field of categorization and creation of semantic annotations for scientific articles

An annotation is a summary of the content of a text, containing the main issues addressed in it [1]. Annotations are classified by content (reference and advisory) and purpose (general and specialized) [7-9]. Reference annotations contain factual information about the document (for example, bibliographic descriptions). Advisory annotations contain a brief assessment of the material and recommend it for a specific category of readers. General concisely conveys the main idea of the document. Specialized contain more detailed information and are aimed at a narrow range of readers. A

separate type is review annotations, which include an analysis of several sources on a specific topic [7-9].

The first use of the term "annotation" dates back to the 2nd century AD, although similar practices existed even earlier. The functional use of annotations dates back to the catalogues of the Library of Alexandria (3rd century BC), where they were used to systematize large volumes of textual information. With the development of information technology and the increase in the volume of textual information, the automation of the annotation process has become an essential task in the field of computational linguistics.

Automatic annotation generation is a difficult task due to difficulties in formalization [10-16]. Natural language has a complex structure, semantic ambiguity and variability of expressions. For effective automation, it is necessary to consider different types of annotations and approaches to their formal representation of the text of the construction.

Let us denote the source text as T and its annotation as A [1]. The creation of an annotation can be represented as the process $T \rightarrow A$, where A contains generalized information from T. The texts can be considered as sets $T = \{t_1, t_2, ..., t_n\}$ (sentences, paragraphs), and the annotation as a subset $A = \{a_1, a_2, ..., a_m\}$, where $\{A\} << \{T\}$ (i.e. the annotation is always much shorter than the source text). The main problem is finding the most important information in the text and presenting it concisely without losing the content. Automatic annotation generation is a difficult task due to difficulties in formalization [15-21]. Natural language has a complex structure, semantic ambiguity and variability of expressions. For effective automation, it is necessary to consider different types of annotations and approaches to their formal representation of the text of the construction.

2.2. Task statement

In the modern world, there is an exponential growth in the number of scientific research, which is becoming extremely important in the context of the development of science and technology. Thousands of new scientific articles and publications in various fields of knowledge appear every year, which creates a large information load for the scientific community. This large volume of scientific literature creates several challenges. First, there is a growing need for tools that help scientists effectively deal with large volumes of data and quickly find the information they need. Second, it is essential to have effective methods of organizing this information for quick and convenient access to it. Third, tools are required in order to help improve the quality of search and analysis, avoiding problems associated with the ambiguity of words or terms. Thus, the emergence of a large number of scientific studies calls into question the relevance and effectiveness of existing methods of organizing and searching for scientific information, which makes the development of new tools and approaches in this field essential.

The purpose of competitor research is to gain a deeper understanding of the competitive environment and identify the key factors that influence the success of our product in the marketplace. This research will give us the opportunity to analyze the functionality, effectiveness and user experience of competing programs, allowing us to identify their strengths and weaknesses. We will be able to identify opportunities to improve our product based on identified weaknesses of competitors and understand what features or capabilities may be key for our users.

The objectives of the study are to consider the main algorithms of generation and extraction to solve the problem of automatic annotation of natural language texts. Based on the priority of semantic analysis among the generation algorithms, please choose a method for modifying it using an approach based on the model of restriction propagation.

2.3. Review of existing solutions

IBM Watson Discovery offers tools for understanding and categorizing textual information, including scientific articles. They use artificial intelligence to analyze content and create semantic annotations to facilitate data retrieval and analysis.

- *Automatic topic recognition*. The platform uses machine learning methods to automatically recognize the topic of text documents. It allows users to quickly identify significant themes and key concepts in datasets.
- *Finding information and grouping results*. IBM Watson Discovery provides powerful search tools that allow users to quickly find the information they need in large volumes of data. It can also group results by topic or other parameters for ease of analysis.
- *Creation of semantic annotations*. The platform can create semantic annotations for text documents that reflect key concepts and relationships between them. It helps users to quickly understand the contents of papers and discover valuable information.
- *Sentiment and sentiment analysis*. IBM Watson Discovery can perform sentiment and sentiment analysis on text documents, detecting the emotion and tone of messages. It can be helpful in identifying trends and sentiments in textual data.
- *Integration with other systems*. The platform can integrate with other systems and tools, which allows users to efficiently and effectively use its functionality in their projects and applications.

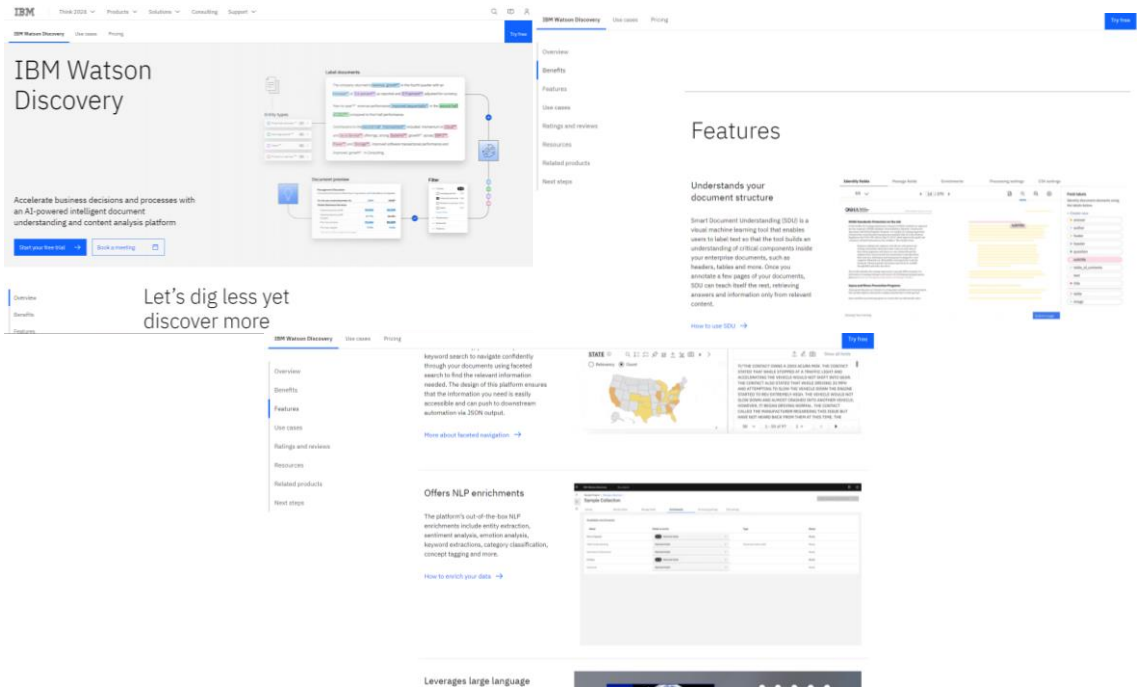


Fig. 1. IBM Watson Discovery offers tools.

Due to the fact that the program is available only with a paid subscription, it was not possible to evaluate its capabilities.

The Semantic Scholar platform specializes in the application of semantic technologies for cataloguing and searching scientific publications. They use advanced natural language processing techniques to create semantic annotations and categorize articles according to various criteria. As you can see, you can search for the necessary scientific articles on given topics.

- *Search for scientific articles*. The platform allows users to search for scientific articles by various criteria, such as author, keywords, subject, journal or specific period.
- *Citation analysis*. Web of Science provides the ability to analyze the citations of scientific articles, which allows you to determine their influence and ranking in scientific circles.
- *Create bibliographic lists*. Users can easily create bibliographic lists for their research and publications.
- Monitoring scientific trends: The platform provides tools for monitoring scientific trends and research development in various fields.
- *Decision support*. Web of Science helps researchers and academics make informed decisions based on an assessment of the scientific literature and its impact.

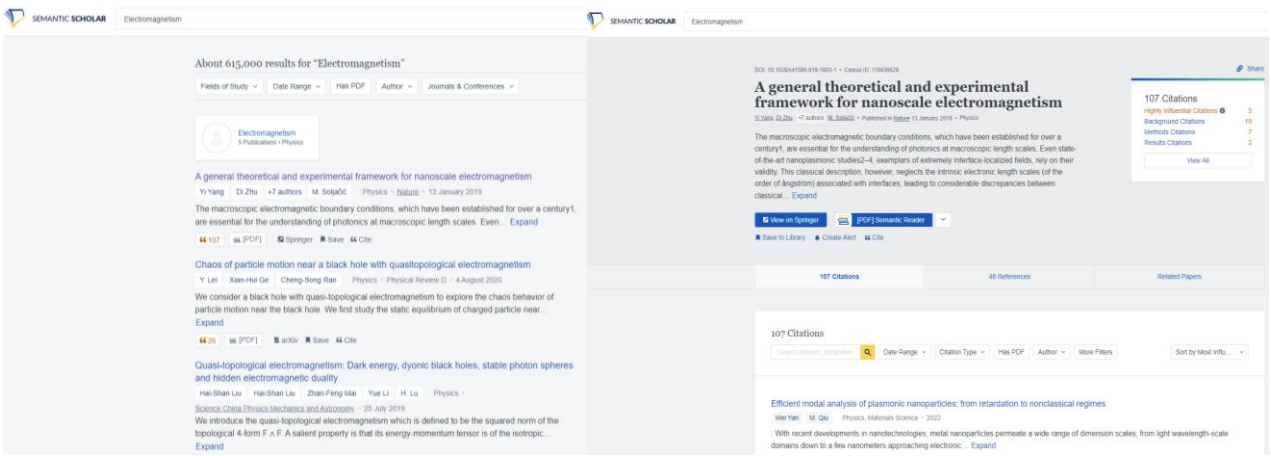


Fig. 2. Semantic Scholar platform

Elsevier's Scopus is a database of scientific publications that provides access to a wide range of scientific journals and conference proceedings. The platform offers tools for semantic analysis and classification of scientific articles. Elsevier's Scopus is one of the largest and most authoritative databases of scientific publications in the world. It contains millions of scientific articles from various disciplines, including natural sciences, social sciences, medicine, engineering and many others. Scopus includes articles from journals, conference proceedings, books and patents, making it an essential tool for scientific research and analysis. The main features of Scopus include:

- *Broad coverage of scientific materials*. Scopus contains a wide range of scientific publications from all branches of science and technology, which allows researchers to get a complete picture of the state of scientific research in their field.
- *Citation analysis*. The platform provides tools for citation analysis, which allows you to determine the influence and importance of scientific publications in the academic environment.
- *Research activity monitoring*. Scopus allows you to track research activity, including publications, citations, h-index and other indicators.
- *Search and filter data*. Users can search and filter data by various criteria, such as author, keywords, subject and other parameters.

Scopus is an essential tool for researchers, universities, libraries and other organizations involved in scholarly research and analysis. It was also not possible to use it due to the fact that its full functionality is available only to those who will publish on it.

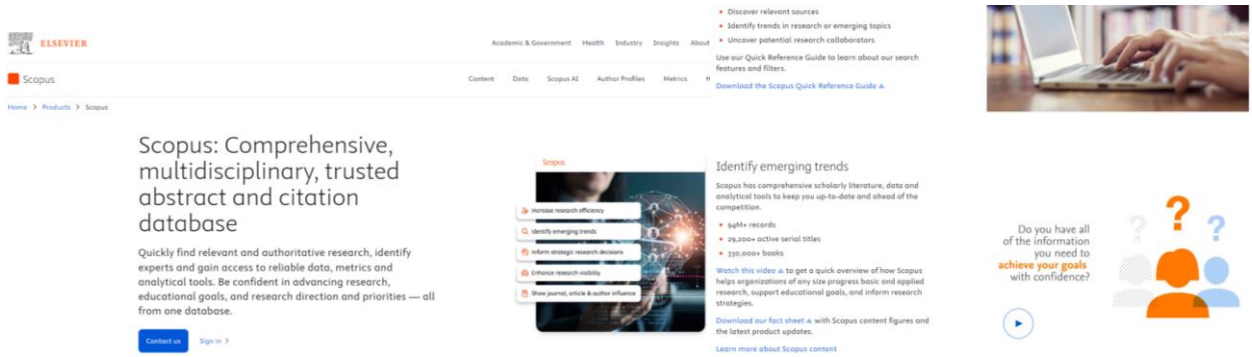


Fig. 3. Semantic Scholar platform

2.4. Advantages of the product under development

Our product has several key advantages that make it unique in the market.

- *Combining categorization and semantic annotations*. We offer an innovative approach that combines the ability to categorize scientific articles with the creation of semantic annotations. It allows users to quickly and efficiently find and understand not only the topic of articles but also their content.
- *Extensibility*. Our product has a flexible architecture that allows you to easily expand its functionality. It means we can quickly introduce new features and improvements based on user needs.
- *Free*. We provide our product for free, making it available to a wide range of users. It is essential for scientific researchers and academic institutions with limited budgets.
- *Idea for further integrations*. Our product opens up vast possibilities for further integrations with other scientific platforms and tools. We are considering opportunities to collaborate with other leading developers of scientific tools to create an even more complete and efficient environment for scientific research.

2.5. Disadvantages of the product under development

Our product, despite its advantages, also has several disadvantages that must be considered.

- *High resource capacity*. The use of our product may require significant computing power resources, especially when processing large volumes of scientific data. It can lead to performance issues and increased hardware costs.
- *Complexity of support*. Due to the complexity of the product architecture and extensibility, support can be demanding and challenging. It can create difficulties in solving problems and providing technical support to users.
- *Difficulty setting*. Customizing our product may require specialized knowledge and skills, which may present problems for new users. A high degree of complexity can result in time and resources being spent on adequately configuring and using the product.

- *Lack of a viewable database as such*. Our product does not yet have a built-in viewable database. It can make it difficult for users to access scientific information and limit their ability to work with data. But this is the first thing that is planned to be added.
- *Primitive user interface*. The interface may not be sufficiently intuitive and convenient for users, which may make it difficult for them to interact with the product and reduce the overall comfort of use. The primitiveness of the interface can also limit the ability of users to use various functions and product options.

2.6. General comparison

We will conduct a general comparison of the product we developed with existing analogues on the market. We will carefully analyse the main parameters, such as the effectiveness of detecting artificially created texts and the accuracy of the results. Based on this comparison, we will be able to find out the advantages and disadvantages of our product compared to competitors, as well as identify its competitive advantages and opportunities for further improvement.

Criteria for evaluating table columns:

- In the Evaluation Criteria column, in the "Ability to create semantic annotations: Yes" and "No" columns,
- The "Free" column will use "Yes" and "No" markers;
- The Extensibility column will use Yes and No markers.

Table 1. The advantages and disadvantages of our product compared to competitors

Competitors	Ability to create semantic annotations	Free of charge	Possibility of expansion
IBM Watson Discovery	Yes	No	Yes
Semantic Scholar	No	Yes	Yes
Elsevier's Scopus	Yes	No	No
IBM Watson Discovery	Yes	No	Yes

IBM Watson Discovery outperforms other systems in semantic analysis capabilities but is expensive and challenging to configure. Scopus has better capabilities for scientific analysis but does not create semantic annotations and is a closed platform. The proposed system provides free access and scalability but requires interface improvements and the addition of a database. Thus, the proposed system has the potential to compete with commercial solutions but needs further improvements for full-fledged use in an academic environment.

Table 2. Comparison of the efficiency of the proposed system with IBM Watson Discovery and Scopus (main features and functionality)

System	Advantages	Disadvantages
IBM Watson Discovery	It uses artificial intelligence to analyze text documents. Performs automatic topic recognition and creates semantic annotations. Includes sentiment analysis of texts to determine the emotional colouring of documents. Integrates with other systems and APIs, allowing you to scale analytical processes.	It is a paid service that limits access to its features.
Elsevier Scopus	Includes millions of scientific publications from journals, conferences, and books; Uses semantic analysis to classify articles; Allows for citation and impact analysis of scientific papers	Full access to the platform is available only to authors and subscribers.
Proposed system	Uses RoBERTa and RAG models for categorization and annotation; Allows for automatic generation of semantic annotations by combining classification with text generation; Free and extensible thanks to its open architecture	The system does not yet have a built-in database and needs interface refinement.

Table 3. Performance comparison

Criterion	IBM Watson Discovery	Scopus	Proposed system
Automatic article categorization	Yes	Yes	Yes
Creating semantic annotations	Yes	Hi	Yes
Free access	No	No	Yes
Expandability	Yes	Hi	Yes
Citation analysis	No	Yes	No
Access Restrictions	Paid subscription	Author access	Open source software
Integration with other platforms	Yes	No	Planned
Flexibility of settings	Complex API	Limited capabilities	Flexible system

2.7. Prospects for product development

Our product opens up broad prospects for further development and improvement. One of the key directions is the constant expansion of functionality, which will allow users more opportunities to create semantic annotations and categorize scientific articles. Improvements in text processing and machine learning algorithms pave the way for improving the quality of results and providing more accurate and relevant annotations. Also, an important aspect is the constant updating and expansion of the database of scientific articles, which will allow our product to remain relevant and competitive. On the other hand, we also plan to focus on implementing integrations with other scientific platforms and services to expand the functionality of our product and ensure it meets the needs of users. Optimizing the user interface is also an important aspect that will allow us to make the use of the product more convenient and intuitive for

users. Overall, these perspectives will allow our product to remain at the forefront of technological development, providing users with the most efficient and convenient tools for working with scientific information.

During the analysis of competitors in the market of categorization systems and the creation of semantic annotations for scientific articles, an urgent problem of effective management of scientific information in conditions of constant growth of data volumes was revealed. The review of existing solutions made it possible to identify their advantages and disadvantages, as well as to identify opportunities for further improvement. Based on this analysis, our product was developed, which combines the capabilities of categorization and creation of semantic annotations for scientific articles, offering a unique approach to the organization and search of scientific information. The advantage of our product is its innovativeness, which consists of the combination of two key functions - categorization and creation of semantic annotations, which allows users to effectively organize and quickly find the necessary scientific information. In addition, our product offers the possibility of expanding functionality and integration with other systems, which makes it flexible and adaptable to the needs of users. However, among the shortcomings, it is worth noting the high resource consumption and complexity of the setting, as well as the primitive interface, which can become an obstacle in using the product.

In general, the results of the analysis of competitors indicate the relevance of our product and its potential in solving the problem of organizing and searching for scientific information. The prospects for its development open up new opportunities for improving the management of scientific data and its implementation in the market of scientific research.

3. Material and Methods

Automatic text annotation methods are divided into two main types: generative algorithms and extraction algorithms. Generative algorithms generate a new text description based on the analysis of the input text. Extraction algorithms select the most relevant fragments of the input text to create an annotation [1-6].

The basic algorithm uses a fragment weight metric that takes into account the number of query words in a fragment and their distance from each other. The most crucial fragment is included in the list of search results. If several fragments have the same weight, the one located closer to the beginning of the text is selected. It has high performance but poorer annotation quality compared to other algorithms.

The Freq Algorithm is an improved version of the basic algorithm that takes into account not only the query words but also the most frequently used words in the document. The fragment weight is defined as a combination of the weight of the basic algorithm and the frequency of keywords. The algorithm improves the quality of the annotation but is inferior to the performance of the basic algorithm.

The LRU-K algorithm is based on the concept of "last recently used". It estimates the local frequency of occurrence of words in a document. Studies have shown that this approach has better annotation quality than Req, as well as significantly higher performance.

The algorithm based on – Semantic Role Labeling (SRL) uses the analysis of semantic argument-predicate structures. It creates annotations through three-level processing: *Syntactic analysis* (Parsing) → *Formation of a dependency tree* → *Lexical construction*. It determines the relationship between words (gender, case, number), which allows for the replacement, reduction and exclusion of words in the text. The disadvantage is the complexity of implementation and the need to accurately determine all relationships between words.

The dataspace-based algorithm uses the approach of integrating disparate information and assessing the reliability of events. It consists of two stages: searching for information about the event and analyzing the reliability using confirmation and refutation coefficients. The main drawback is the lack of a specific method for generating annotations based on the obtained coefficients.

The algorithm based on Text Relationship Maps (TRM) formalizes the text in the form of a graph. The vertices of the graph are fragments of text represented as weight vectors. The edges of the graph reflect the degree of similarity between the fragments, determined using the scalar product of vectors. The annotation is created by selecting the most connected vertices. Advantages: allows for semantic analysis and clustering of texts by topic. The disadvantage is the complexity of building a map of text relationships.

The algorithm, which uses generalized parsing and structural transformations TXL, is based on the use of the TXL language (a programming language for analyzing and transforming texts). The annotation process consists of three stages:

1. Analysis of the input text (obtaining structured information).
2. Designation of semantic categories (selection of words with positive and negative features).
3. Generating an annotation in XML format for further use in databases.

Disadvantage: requires manual correction at the final stage, which makes this algorithm semi-automatic.

The fastest is the basic algorithm, but it has a lower quality of annotations. SRL, TRM and TXL provide the best quality of annotation, but they have a high complexity of implementation. The Dataspace-based algorithm can be helpful in assessing the reliability of information. LRU-K provides the optimal balance between quality and

performance. Thus, the choice of algorithm depends on the needs:

- If fast processing is required, it is better to use LRU-K or the basic algorithm.
- If semantic accuracy is essential, it is advisable to use SRL or TRM.
- If information verification is required, it is worth using the Dataspace-based algorithm.

3.1. Automatic Annotation Methods

Let us consider two main approaches to automatic text annotation: extraction algorithms and generating algorithms (Fig. 4). Extraction algorithms select the most significant fragments of the text and form an annotation by combining these fragments. They use statistical and lexical criteria, such as the frequency of occurrence of keywords or proximity to the beginning of the text. Advantage: speed and low computational costs. Disadvantage: loss of semantic connections, possible duplication of information.

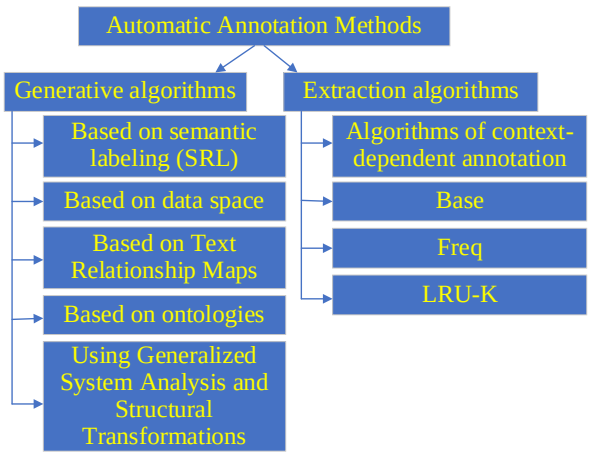


Fig. 4. Classification of automatic annotation methods

Generating algorithms analyze the input text and create a new annotation text based on an understanding of its content. They include deep semantic analysis, which provides more informative and concise annotations. Advantage: accuracy and absence of duplications. Disadvantage: high computational costs and complexity of implementation.

Formally, the annotation process can be represented as a function $T \rightarrow A$, where T is the input text (a set of sentences, paragraphs, or words), A is the output annotation containing the concise content of the text T. Since the annotation is a shortened version of the text, the inequality holds: $|A| \ll |T|$ i.e. the size of the annotation is much smaller than the size of the source text. Extraction methods use fragment weight estimation methods to select the most critical parts of the text. The formula for estimating the weight of a fragment looks like this:

$$W_f = \sum_{i=1}^{n} w_i - \frac{d}{m},$$

where W_f is the total weight of the fragment, w_i is the weight of the i-th query word in the fragment, d is the distance between the first and last query word, m is the total number of queried words in the fragment.

This formula allows us to determine relevant fragments by summing the weights of keywords and taking into account the distance between words to favour compact fragments. For example, consider the text:

"Методи глибокого навчання широко застосовуються для автоматичної обробки природної мови." ["Deep learning methods are widely used for automatic natural language processing."]

Let the words "автоматична" ["automatic"] and "обробка" ["processing"] be part of the query:

- $w_1 = 3.2$ (weight of the word "автоматична" ["automatic"]),
- $w_2 = 2.8$ (weight of the word "обробка" ["processing"]),
- $d = 5$ (distance between words in the sentence),
- $m = 2$ (the query contains two words).

Then $W_f = (3.2 + 2.8) - \frac{5}{2} = 3.2 + 2.8 - 2.5 = 3.5$. Since $W_f > 3$, this fragment is considered relevant and can be included in the annotation.

Choosing an annotation method:

- If speed is the primary goal, it is worth using extraction methods, such as LRU-K or the Freq algorithm.
- If the deep learning of the text is essential, it is better to use generative algorithms, such as TXL or TRM.

Suppose you need to create an annotation for a large number of news articles:

- Extraction method (LRU-K) - selects the most repeated fragments, which allows you to quickly process a large number of texts.
- The generation method (TRM) creates unique generalizations that are better suited for the in-depth analysis of individual articles.

Extraction methods are faster but less accurate (effective for large amounts of data). Generation methods - give better results but have high computational costs. The formula for calculating the weight of a fragment allows you to effectively determine the most significant parts of the text. The choice of method depends on the volume of the text, the quality of the annotation, and computing capabilities. Among the extraction algorithms, the most popular are the algorithms for context-sensitive annotation of HTML documents. Search engines use annotations compiled by similar methods to describe the results in the form of short, continuous pieces of text according to the user's request. The choice of the optimal fragment of text is carried out on the basis of calculating the weights of fragments.

3.2. Basic algorithm

The basic algorithm of automatic annotation is aimed at selecting the most relevant fragments of text for forming an annotation. It is based on calculating the weight of a fragment using the frequency of occurrence of keywords and their location in the text. The algorithm consists of the following stages:

1. Calculating the weight of words in the text.
2. Calculating the weight of a fragment based on the frequency of keywords and their distance.
3. Selecting the most relevant fragment for forming an annotation.

For each word in the query, its weight is determined by the formula:

$$W_i = \log \frac{N}{n_i}, \tag{1}$$

where W_i is the weight of the i-th word in the query, N is the total number of documents in the collection, n_i is the number of documents in which the i-th word occurs. This formula is based on the inverse document frequency (IDF), which is used in search algorithms such as TF-IDF. The weight of a text fragment is calculated using the formula:

$$W_f = \sum_{i=1}^{m} W_i - \frac{d}{m}, \tag{2}$$

where W_f is the total weight of the fragment, W_i is the weight of the i-th keyword in the fragment, d is the distance between the first and last occurrence of keywords, m is the number of keywords in the fragment. This formula allows you to:

1. Prioritize fragments with a large number of essential words.
2. Give preference to fragments in which the keywords are located close to each other.

Fragment selection rule:

- If two fragments have the same weight, the one located closer to the beginning of the text is selected.
- The most crucial fragment is added to the list of results.

An example of how the algorithm works:

- Input text: "Методи машинного навчання широко використовуються для аналізу текстів і створення автоматичних анотацій." ["Machine learning methods are widely used to analyze texts and create automatic annotations."]
- Query: The user is looking for information about "машинне навчання" ["machine learning"] and "автоматичні анотації" ["automatic annotations"].
- Word weight calculation: $W_{\text{learning}} = \log \frac{10000}{200} = 1.7$ and $W_{\text{automatic}} = \log \frac{10000}{500} = 1.3$.
- Fragment weight calculation. Total word weight: $1.7 + 1.3 = 3.0$. Distance between keywords is $d = 5$, $m = 2$

Fragment weight: $W_f = 3.0 - \frac{5}{2} = 3.0 - 2.5 = 0.5$. Since this fragment has the highest weight, it will be selected for annotation. The basic algorithm is simple and fast, which makes it practical for extensive text collections. The formula takes into account the weight of keywords and their location in the text, which allows you to select the most relevant fragments. Despite the high-performance, the quality of annotations is inferior to that of more complex algorithms such as Req, LRU-K, or SRL.

Table 4. Advantages and disadvantages of the algorithm

N	Advantages	Disadvantages
1	Fast execution - the basic algorithm is one of the most efficient in terms of performance.	Low quality of annotation - compared to more complex algorithms (for example, SRL or TRM), the basic algorithm creates less meaningful annotations.
2	Minimal computational costs - does not require complex semantic analysis.	Loss of context – the algorithm does not take into account the meaning of words in sentences, which can lead to the inclusion of uninformative fragments.

3.3. Req Algorithm

Unlike the basic algorithm, the Req Algorithm adds weight to the most repeated words in the text, which allows you to more accurately highlight essential fragments. The formula for calculating the weight of a text fragment:

$$W_{req} = W_{base} + \sum_{i=1}^{n} f_i \quad (3)$$

where W_{req} is the improved weight of the text fragment, W_{base} is the weight calculated by the basic algorithm (taking into account the words in the query), f_i is the frequency of occurrence of the i-th word in the document, n is the number of the most frequent words in the fragment.

Thus, if a specific text fragment contains many words with a high frequency of occurrence, its value will be increased.

Table 5. Argument for using the Req Algorithm

N	Advantages over the basic algorithm	Disadvantages
1	Taking into account the importance of frequently repeated words which makes annotations more accurate.	Requires additional calculations to calculate word frequencies.
2	Improved definition of key text fragments	Loses in performance to the base algorithm

Example of the algorithm. Input text (fragment of a scientific article): "Machine learning models are widely used in scientific research. These models enable automatic categorization of articles and improve the efficiency of data analysis." Processing steps:

1. We highlight the most frequent words in the text. For example: "models" – 2 times, "scientific" – 1 time, "data" – 1 time.
2. We calculate the weight using the base algorithm ($W_{base} = 5.2$).
3. We add the weight of the most frequent words ($f_1 = 2, f_2 = 1, f_3 = 1$): $W_{req} = 5.2 + (2 + 1 + 1) = 9.2$. The fragment with a weight of 9.2 will be considered more important for an annotation since it contains keywords that occur frequently in the document.

The Req Algorithm improves the quality of annotation by taking into account frequent words in the document. It is used in automatic text summarization systems. It has a higher computational complexity compared to the basic algorithm. This algorithm can be effectively used in combination with other NLP methods to create semantically enriched annotations.

3.4. LRU-K algorithm

Unlike previous algorithms (Base Algorithm, Req Algorithm), LRU-K takes into account the frequency of occurrence of a word in the local context and not in the entire text. The formula for calculating the weight of a text fragment in LRU-K:

$$W_{lru} = \sum_{i=1}^{n} f_i \cdot \left(1 - \frac{r_i}{R}\right),$$

where W_{lru} is the improved weight of the text fragment, f_i is the frequency of occurrence of the i-th word in the fragment, r_i is the position of the last use of the word in the text, R is the total length of the text.

Thus, words that have appeared recently have a greater weight than words that have appeared earlier.

Table 6. Argument for using LRU-K

N	Advantages over other algorithms	Disadvantages
1	Taking into account the local context and the last use of the word.	Higher computational complexity due to the need to track the positions of words
2	Improved performance compared to Req Algorithm	It performs worse for static texts, where all words are equally important.
3	Better annotation quality for dynamic texts	

Example of the algorithm

- Input text (fragment of a scientific article) "Deep learning methods have been successfully applied in various fields, including natural language processing and image recognition. These methods utilize artificial neural networks to extract meaningful features from data."
- Processing steps:
    1. Counting word frequencies in the local context: "methods" – 2 times, "deep" – 1 time, "neural" – 1 time.
    2. Calculating the weight taking into account the last use of the word.
        - $r_{methods} = 10, r_{deep} = 2, r_{neural} = 18$;
        - $R = 20$ (text length);
        - $W_{lru} = \left(2 \cdot \left(1 - \frac{10}{20}\right)\right) + \left(1 \cdot \left(1 - \frac{2}{20}\right)\right) + \left(1 \cdot \left(1 - \frac{18}{20}\right)\right)$;
        - $W_{lru} = 2(0.5) + 1(0.9) + 1(0.1) = 1 + 0.9 + 0.1 = 2.0$.

A fragment with a weight of 2.0 will be considered more important in the annotation process. LRU-K Algorithm improves the quality of annotation by taking into account the last use of words. Effective for context-sensitive text categorization. Requires additional resources to track word positions in the text. This algorithm can be effectively used in search engines and automatic text annotation systems, particularly for dynamic texts that change over time.

3.5. Algorithm based on semantic labelling (SRL)

Unlike other methods, SRL focuses on semantic analysis of the text, highlighting predicates, arguments and connections between them. The three-stage markup process:

1. Parsing – parsing a sentence to identify parts of speech and grammatical dependencies.
2. Dependency Tree Construction – creating a graph of connections between words to determine their roles.
3. Lexical Construction – forming an annotation by substituting and reducing semantically essential elements.

For each text fragment T, the algorithm finds a set of predicates P and arguments A, determining their semantic correspondence:

$$A = f(T) = \{(p_1, a_{11}, a_{12}), (p_2, a_{21}, a_{22})\}$$

where p_i – predicates (main actions or events in the text), a_{ij} – arguments (nouns that depend on predicates).

Example of sentence processing:

1. Input text: "The scientist analyzed the dataset and published the results."
2. SRL markup:
    - Predicates analyzed and published;
    - Arguments: analyzed → (Who? scientist, What? dataset); published → (Who? scientist, What? results);
    - SRL annotation: "A scientist analyzed the dataset and reported the findings."

Table 7. Argument for using SRL

N	Advantages of SRL	Disadvantages of SRL
1	Avoidance of unnecessary repetitions – unnecessary parts of the text are not copied in the annotation.	The complexity of implementation – requires a grammar analyzer and methods for processing syntactic relations.
2	Deep semantic analysis takes into account not just the frequency of words but also their role in the text.	High computational costs – compared to simple algorithms, SRL requires more resources.
3	Structured annotation construction – results are more straightforward to perceive and generalize.	

Table 8. Comparison with other algorithms

Criteria	Base algorithm	Req Algorithm	LRU-K Algorithm	SRL Algorithm
Analysis method	Frequency	Frequency+context	Last Activity	Semantic parsing
Context considerations	None	Partial	Partial	Full
Annotation quality	Average	Higher	Higher	High
Computational complexity	Low	Average	Average	High

SRL algorithm provides the highest quality annotations, as it takes into account semantic dependencies in the text. It is used in complex NLP systems for automated analysis of scientific texts and their generalization. It requires significant computational resources so that it can be implemented in combination with other methods (for example, Transformer models). This algorithm is handy for automatic referencing of texts, academic research, and annotation of large text corpora.

3.6. Dataspace-based algorithm

Dataspace-based Algorithm consists of two main stages:

1. Integration of disparate data and retrieval of information about the event
2. Annotation generation and calculation of confirmation and refutation coefficients

Dataspace is a structure that includes databases and repositories (SQL, NoSQL, static web pages), local indexes and repositories (for quick access to pre-processed texts) and information search, processing and integration tools (for extracting relevant information). An adaptive media ontology is used, which allows for a quantitative assessment of confirmation and refutation coefficients. Based on the obtained values, an annotation is built, which consists of two paragraphs: the first confirms the event under consideration, and the second refutes it. The formula for calculating the coefficient of confirmation (C_{conf}) and refutation (C_{ref}):

$$C_{conf} = \frac{\sum_{i=1}^{N} w_i \cdot R_i}{\sum_{i=1}^{N} w_i} \text{ and } C_{ref} = \frac{\sum_{i=1}^{M} w_i \cdot (1-R_i)}{\sum_{i=1}^{M} w_i},$$

where w_i is the weight of the information source, R_i is the source relevance score (from 0 to 1), N is the number of confirming sources, M is the number of refuting sources. If $C_{conf} > C_{ref}$, then the event is considered confirmed, and vice versa.

Table 9. Argument for using Dataspace-based Algorithm

N	Advantages	Disadvantages
1	Combining information from different sources – improves the quality of the annotation.	High requirements for computing resources – requires fast search and analysis of a large number of sources.
2	Objective assessment of events – due to the balance between confirmation and refutation.	The problem of assessing the reliability of sources is the need for an ontological model for the automatic determination of reliability.
3	Flexibility in working with texts of different structures (news, scientific articles, social networks)	

Example of the algorithm. The task is the automatic annotation of the event "Recent studies confirm the effectiveness of quantum computers". Data sources: Scopus database (10 articles), news portals (15 articles) and social networks (50 posts). Calculation of coefficients:

- 8 out of 10 Scopus articles, 12 out of 15 news materials, and 30 out of 50 posts confirm.
- Refutes: 2 Scopus articles, three news materials, 20 posts:

$$C_{conf} = \frac{(8 \cdot 0.9)+(12 \cdot 0.7)+(30 \cdot 0.5)}{8+12+30} = 0.68 \text{ and } C_{ref} = \frac{(2 \cdot 0.9)+(3 \cdot 0.7)+(20 \cdot 0.5)}{2+3+20} = 0.52.$$

The value of $C_{conf} > C_{ref}$. Therefore, the event is considered confirmed. A dataspace-based Algorithm allows you to evaluate events based on large data sets and uses adaptive ontology to analyze the reliability of sources. The method is suitable for scientific research, fact-checking and social network analysis. The main challenge is the need for significant computing resources and the development of a high-quality model of trust in sources. This algorithm can be effectively used in journalism, scientific research

3.7. Algorithm based on text relationship maps

The main idea of the TRM (Text Relationship Maps) algorithm is to formalize the text as a graph, where the vertices correspond to fragments of the text, and the edges denote the semantic relationship between the fragments. Let us describe the graph model of the text. The text is represented as a graph:

$$G = (V, E), \tag{4}$$

where V is a set of vertices, each of which corresponds to a fragment of text, E is a set of edges indicating the connections between text fragments. Each vertex is represented as a vector description containing the weights of keywords:

$$V_i = (w_1, w_2, \dots, w_n) \tag{5}$$

where w_j are the weights of words in a text fragment. Edges connect vertices with a high degree of similarity, which is calculated through the scalar product of the vectors of the corresponding fragments. Calculation of semantic proximity between text fragments:

$$S(V_i, V_j) = \sum_{k=1}^{n} w_{ik} \cdot w_{jk}, \tag{6}$$

If the value of $S(V_i, V_j)$ exceeds a certain threshold θ, then a connection is established between the fragments. The more connections (edges) a vertex has, the more critical the corresponding fragment is. The selection of annotation fragments is carried out by sorting the vertices by the number of connections.

The weight of each vertex v_i is calculated by the formula:

$$W(v_i) = \sum_{j=1}^{m} f_j \cdot w_j,$$

where f_j is the frequency of occurrence of word j in fragment v_i, w_j is the importance of the word j (determined by TF-IDF or other methods). The connections between text fragments are formed using the scalar product of fragment vectors:

$$S(v_i, v_j) = \frac{v_i \cdot v_j}{|v_i||v_j|},$$

where v_i, v_j – vectors of text fragments obtained using word2vec, BERT or TF-IDF, $S(v_i, v_j)$ is a measure of similarity between fragments. The higher the value of $S(v_i, v_j)$, the greater the probability that the fragments have a common context.

Table 10 Argument for using TRM

N	Advantages	Disadvantages
1	Deeper semantic analysis provides better annotation relevance.	High computational cost – an efficient similarity calculation mechanism is required.
2	Ability to cluster documents by topic	The problem of setting the correct similarity threshold θ.
3	Increased accuracy of searching for similar documents	

Example 1 of the algorithm:

- Input text (fragment of a scientific article) "Machine learning is transforming various industries. Neural networks allow the automation of complex tasks. Data-driven models are improving decision-making."
- Steps to build a graph:

1. Breaking text into fragments:

$V_1 \rightarrow$ *Machine learning is transforming various industries.*
$V_2 \rightarrow$ *Neural networks allow automation of complex tasks.*
$V_3 \rightarrow$ *Data-driven models are improving decision-making.*

2. Calculating semantic similarity between fragments:

$S(V_1, V_2) = 0.85 \rightarrow$ Connection established.
$S(V_2, V_3) = 0.92 \rightarrow$ Connection established.
$S(V_1, V_3) = 0.45 \rightarrow$ connection not established (below threshold $\theta = 0.5$).

The most crucial fragment is V_2, since it has two connections. The annotation will consist of the most connected fragments.

Example 2 of the algorithm. Input text: "Machine learning models are widely used in research. These models improve the efficiency of data analysis. Deep learning techniques allow extracting meaningful patterns from large datasets."

Table 11. Graph construction

Vertex	Text fragment	Top weight $W(v)$
v_1	Machine learning models are widely used in research.	1.5
v_2	These models improve the efficiency of data analysis.	1.3
v_3	Deep learning techniques allow extracting patterns.	1.8

Calculating similarity between fragments: $S(v_1, v_2) = 0.76, \quad S(v_2, v_3) = 0.64, \quad S(v_1, v_3) = 0.50$. Based on these values, a graph is formed, where the most connected vertices determine the primary annotation of the text. Generated annotation: "Machine learning and deep learning techniques significantly enhance data analysis efficiency."

TRM allows you to create semantically relevant annotations by analysing textual relationships. The method is used to cluster documents and improve the search for similar texts. TRM will enable you to structure the text and highlight key relationships between fragments. It requires significant computational resources, which makes it less practical for vast text corpora. It has high computational complexity, so it needs optimization to work with large text corpora. This algorithm can be effectively used in library systems, search engines, and scientific databases. This approach is practical for automatic annotation of significant scientific texts and articles, especially in cases where it is necessary to highlight relationships between disparate text fragments.

3.8. Algorithm using generalized parsing and structural transformations of the TXL system

TXL (Tree Transformation Language) is a specialized programming language designed for software analysis and text document transformation. In this case, TXL is used for semantic annotation of texts, using generalized parsing and structural transformations. The algorithm consists of three main stages:

1. Text parsing using TXL to obtain an approximate sentence structure;
2. Identification of positive and negative indicators of semantic categories to create an initial annotation;
3. Saving the XML markup in a database for further analysis.

The formula for constructing the semantic structure of a text $S = \{s_1, s_2, \dots, s_n\}$, where S is the set of sentences in the text, s_i is a single sentence that passes through the TXL processor to analyze its semantic content.

Definition of significant words using TXL:

$$W_{TXL} = \sum_{i=1}^{n} w_i \cdot r_i,$$

where w_i is word weight according to semantic category, r_i is the frequency of occurrence of the word in the text.

XML markup after analysis:

```
xml
CopyEdit
<annotation>
<sentence id="1"> The research applies deep learning techniques. </sentence>
<category type="positive"> deep learning </category>
<category type="neutral"> research </category>
</annotation>
```

Table 12. Argument for using the TXL algorithm

N	Advantages of TXL	Disadvantages of TXL
1	Ability to perform generalized text analysis — TXL can work with scientific articles, technical documents, and other types of texts;	Requires expert correction at the final stage, which complicates full automation.
2	Automatic structuring of information through XML format;	High requirements for computing resources when analyzing large text corpora
3	Flexibility in changing parsing and categorization rules	

Example of TXL algorithm operation:

- Input text: "Neural networks are widely used in modern artificial intelligence applications."
- Processing process:
  1. Parsing using TXL (syntactic structures of the sentence are determined).
  2. Semantic categorization:
     a. "Neural networks" → category "positive".
     b. "artificial intelligence" → category "neutral".
  3. Saving in XML format:

```
xml
CopyEdit
<annotation>
<sentence id="1"> Neural networks are widely used in modern artificial intelligence applications.
</sentence>
<category type="positive"> neural networks </category>
<category type="neutral"> artificial intelligence </category>
</annotation>
```

A semantic annotation of the text is generated, which can be used to categorize or create a document summary.

The TXL algorithm is an effective method of semantic analysis that allows you to automate the processing of text documents. XML markup facilitates integration with databases and other text analysis systems. It requires manual refinement, which may limit its full automation. Development prospects:

1. Using GPT models in combination with TXL for automatic correction of annotations;
2. Optimization of the algorithm for parallel processing of large text arrays.

The TXL algorithm can be effectively used in scientific article analysis systems, search engines, and automatic document categorization.

3.9. System analysis of the research object and subject area.

The object of research is the process of categorization and creation of semantic annotations for scientific articles. This process includes the systematization of scientific materials according to various criteria and the selection of key terms, concepts and conclusions in each article. Categorization helps organize articles according to multiple aspects such as subject matter, research methodology, scope, etc., thereby creating a structured database of scientific information. Semantic annotations, in turn, provide short descriptions that reflect the main aspects of articles and help in their search and understanding.

The subject area is scientific articles and their information environment. Scientific articles are the main object of analysis, as they are a means of transmitting scientific information and research results. The information environment includes databases of scientific journals, websites of scientific publications, conference archives, bibliographic services and other sources where scientific data is stored and distributed. The structure and content of scientific articles are also a component of the subject area, as they contain key terms, concepts and other elements that are subject to categorization and annotation. The object of research is a complex set of methods and approaches aimed at identifying and analysing textual materials that were generated by artificial intelligence systems, in particular chatbots. One of the key tasks in this context is to distinguish such texts from those created by real users. The research includes the development and application of machine learning algorithms, the creation of models for the analysis of text data, and the development of other tools for the effective detection and processing of these texts. The main focus is on the development of approaches that will allow improving the process of detection and identification of texts created by artificial intelligent agents.

3.10. Tree of objectives.

The general goal of our research work is the development and implementation of a system that will provide automated detection of such texts. It is an essential step in ensuring the quality and reliability of information on the Internet, as well as in the fight against manipulative information. The general purpose is to develop a system that will ensure the creation of semantic annotations and categorization of scientific articles. Aspects of the general goal:

- *Accuracy of article categorization.* Evaluates how effectively the system can classify scientific articles according to various criteria, such as subject matter, research methodology, scope, etc.
- *Effectiveness of creating semantic annotations*. Evaluates how accurately and informatively the system can generate short descriptions of key aspects of scientific articles.
- *Adaptability to changes in the structure of articles* is evaluated by the system's ability to adapt to changes in the structure and format of scientific articles that may change over time.
- *Recognition of language features*. Evaluate how well the system can take into account language differences and the specifics of various scientific disciplines for more accurate categorization and creation of semantic annotations.

System performance quality criteria:

- *Categorization accuracy.* Measured by the percentage of correctly classified articles relative to the total number processed.
- *The quality of semantic annotations* is measured by the degree of correspondence of semantic annotations to the content of articles and their key aspects.
- *The flexibility of the system* is evaluated by the ability of the system to adapt to changes in the format and structure of scientific data.
- *Speed*. Measured by the time required for categorization and creation of semantic annotations for scientific articles.

3.11. Building a UML diagram of use cases.

When developing software or other information systems, an important step is to find out, analyze and document what the system should do. Using Use Cases in the Unified Modeling Language (UML) is an effective way to do this. They allow us to describe how the system communicates with users or external agents to achieve certain goals or perform specific tasks.

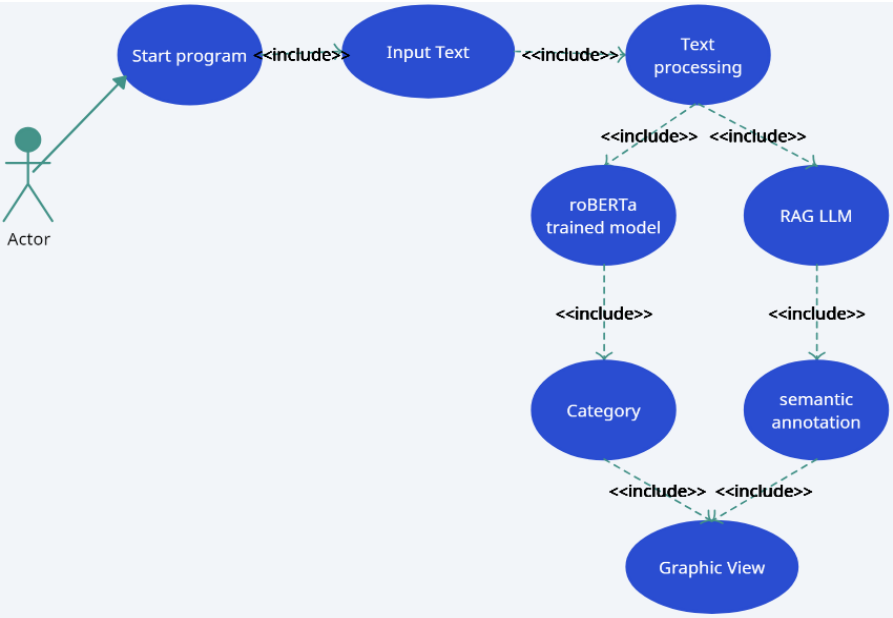


Fig. 5. UML diagram of use cases

3.12. Machine Learning Methods in Research: Constraint Propagation Models and NLP

The study presents the use of Constraint Propagation Models and Natural Language Processing (NLP) methods for automatic categorization and semantic annotation of scientific articles.

1. Constraint Propagation Models. Constraint propagation is a problem-solving method that uses a system of rules and constraints to find the optimal solution. This approach is widely used in artificial intelligence, pattern recognition, and text analysis. In this study, the constraint propagation model is used to identify the most relevant terms in the text, construct a graph of relationships between words, and categorize scientific articles by analyzing semantic relationships. Text fragments are represented as vertices of a graph, and the relationships between them are described as edges $G = (V, E)$, where V is the set of vertices (keywords or phrases), E is the set of relationships between vertices. The number of connections to other terms determines the weight of a vertex.

Table 13. Comparison of tools

N	Name	Advantages	Disadvantages
1	Constraint Propagation Models	High flexibility when working with different texts. Possibility of automatic categorization without prior training on a sample.	High computational cost for graph construction. Difficulty in scaling for unstructured data.
2	RoBERTa	Higher accuracy compared to BERT. Better consider context in long texts.	High computational cost.
3	RAG	Higher relevance of annotations thanks to database access; Flexibility in combination with different knowledge sources.	High resource costs; Requires high-quality knowledge base setup

2. Natural Language Processing (NLP) Methods in the Study

The study used modern NLP methods for automatic text processing. In particular, the RoBERTa and RAG models were used.

2.1. Features of the RoBERTa (Robustly Optimized BERT Pretraining Approach) model.

• An improved version of BERT (extended based on a large corpus of data).

• Uses Byte Pair Encoding (BPE) for tokenization.

• Shows better results in text categorization than traditional BERT.

Each article is converted into a vector $X = \{x_1, x_2, \dots, x_n\}$. RoBERTa determines the probability of belonging to each category:

$$P(y_k|X) = \frac{e^{W_k \cdot X}}{\sum_j e^{W_j \cdot X}},$$

where W_k is model weighting factors.

2.2. RAG (Retrieval-Augmented Generation) combines a generative model and a knowledge base to generate answers.

• Allows you to create annotations based on external sources.

• Uses a search for relevant information before generation.

RAG structure:

• Retrieval (Context Search) - finds articles that are related to the query.
• Generation (Text Generation) - uses the found documents to create an annotation.

$$P(A|T) = \sum_D P(A|T, D)P(D|T),$$

where A is annotation, T is source text, D is retrieved document.

Table 14. Comparison with similar methods

Method	Use in research	Alternatives	Main differences
Constraint Propagation Models	Text structure analysis, categorization	Latent Dirichlet Allocation (LDA)	Flexibility in defining relationships between words
RoBERTa	Classification of articles	BERT, XLNet	Improved context awareness, better category prediction
RAG	Annotation generation	T5, GPT-3	Allows you to search for additional information before generating text

The study uses a combination of constraint propagation and NLP, which provides high accuracy in categorization and annotation. RoBERTa significantly outperforms standard BERT models, providing better context awareness. RAG allows for the creation of semantically enriched annotations by combining search and generation. For further improvement, one can consider the T5 or GPT-4 models, which show better results in text generation.

3.13. Statement and justification of the problem

Purpose and place of application of the system. Considering the importance of scientific research in the modern world, the system of creating semantic annotations and categorization of scientific articles has an essential purpose in the organization and analysis of scientific information. The main goal of this system is to facilitate access to scientific knowledge by automating the process of organization and structuring of scientific materials. The purpose of the system is:

- *Automatic selection of key ideas*. The system uses modern methods of natural language processing to identify and highlight key topics, terms and concepts in scientific articles. It allows researchers to quickly navigate the volume of information and find the necessary materials for their research.
- *Categorization and organization of information*. The system distributes scientific articles by thematic categories or keywords, which allows users to quickly find the information required on a specific topic or research area.

The place of application of the system is divided into various spheres of scientific activity and education:

- *For scientific researchers*. The system helps scientists find the necessary information for their research faster and more efficiently, as well as analyse and compare scientific data.
- *For universities and scientific institutes*. The system can be used to support library funds and the organization of scientific events and research projects.
- *For publishers.* The system contributes to the improvement of the quality of peer review of scientific articles and the management of scientific journals, contributing to the dissemination and development of scientific information.

The application of the system has the potential to significantly improve the efficiency of scientific research and make scientific information more accessible and understandable to a wide range of users.

The development of a conceptual model of the system for creating semantic annotations and categorization of scientific articles is a key stage in the software design process. This model defines the main components, their relationships and system functionality, taking into account the needs of users and the specifics of scientific information processing. The target audience of the system includes scientists, students, librarians and other specialists who are engaged in searching, analysing and organizing scientific data. The main functions of the system will consist of the creation of semantic annotations for scientific articles and their subsequent categorization according to various criteria.

The conceptual model of the system will consist of the following components:

1) User interface:
- Navigation menu for convenient access to various system functions;
- Forms for entering search queries and displaying results;
- Tools for editing and managing annotations;

2) Natural language processing module. Using natural language processing algorithms to analyse and extract key terms and concepts from the text of scientific articles.

3) Categorization module:

- Automatic or semi-automatic categorization of scientific articles by topic, authors, keywords, etc.;
- Ability to create and edit your categories;

4) Database:

- Storage of scientific articles together with their semantic annotations and category information;
- Providing access to data and updating it;

5) Access control module. Setting user access rights to system functions and data.

This conceptual model of the system envisages the development of an integrated environment that will help scientists and researchers effectively manage and analyse scientific information for further use in their research and publications.

As part of this study, the following was carried out:

- *Analysis of the object of research and subject area* revealed the main object of research - the process of categorization and creation of semantic annotations for scientific articles. The importance of this process for the organization and analysis of scientific information is emphasized.
- *Definition of the goal tree*. Based on the analysis, the general goal of the project was formulated - to develop a system that will provide automated creation of semantic annotations and categorization of scientific articles. Aspects of this goal were also identified, including article categorization accuracy, semantic annotation efficiency, adaptability to changes in article structure, and language feature recognition.
- *Constructing a UML Use Case Diagram*. Using the UML methodology, a use case diagram was developed to describe how the system communicates with users or external agents to achieve specific goals. It is a tool for documenting system functionality requirements.
- *Statement and substantiation of the problem*. The purpose and place of application of the system of creating semantic annotations and categorization of scientific articles were considered. It was determined that the system is aimed at facilitating access to scientific knowledge and improving the organization and structuring of scientific materials.
- *Development of a conceptual model of the system*. A conceptual model of the system was developed, including a user interface, a natural language processing module, a categorization module, a database and an access control module. This model defines the main components of the system and their functionality, taking into account the needs of users and the specifics of scientific information processing.

3.14. Selection and justification of methods for solving the task

For the system to work effectively, practical methods and tools are needed to help them distinguish between texts and make appropriate decisions. In this context, it is essential to choose and justify the methods and means of presenting knowledge in the decision-making system, as well as the description of the mechanisms of logical deduction in the system and the justification of decision-making algorithms. The methods and tools used in the system that detects texts written by artificial intelligence are considered and substantiated here. Algorithms used to classify such texts and make appropriate decisions are also described and substantiated:

- *Text tokenization* in the roBERTa model is performed using a unique technique that differs from traditional methods. roBERTa uses Byte Pair Encoding (BPE) tokenization, which is an effective strategy for working with text.
- *Text vectorization* in the roBERTa model consolidates information about each token in the text, taking into account its semantics, position, and context. It is good because this complex vectorization allows the model to understand the text more deeply and consider its full context when making decisions, resulting in more accurate and adaptable results.
- *Training data* is a set of data used to train a machine learning or deep learning model. This set consists of example inputs (functions) and their corresponding outputs (labels or target values). During training, the model uses this data to find the optimal parameters that best represent the relationships between the input and output data, with the goal of later using it to predict new data. Training data determines the quality and efficiency of the model when working with actual data.
- *Using ready-made embeddings* is a practical approach in the field of natural language processing. It makes it possible to effectively use the semantic information embedded in the models during their training on large text corpora without the need to spend time and resources on training the model from scratch. Ready-made embeddings are generic, save time and computing resources, and are available for a wide range of tasks, from text classification to machine translation. Using ready-made plugins allows you to improve the quality and speed of text data processing in your application or study.
- *Using off-the-shelf Large Language Models (LLMs) with Retrieval-Augmented Generation (RAG)* is a powerful approach in the field of natural language processing. RAG combines the capabilities of LLM with the ability to interact with a database to obtain context-sensitive answers. This approach makes it possible to create systems that are able to generate answers to questions using a broad context of information. The use of ready-made LLM with RAG provides flexibility and efficiency in solving various text-processing tasks. It

offers the opportunity to create solutions that adapt to the needs of users using information from available data sources.

- *The use of performance evaluation metrics* such as loss and accuracy is an essential aspect in training and evaluating machine learning models. Losses reflect the model's error rate during training, where low values indicate a better fit of the model to the data. Accuracy measures the percentage of correctly classified examples, and it is a key metric for evaluating performance in classification problems. Using these metrics allows you to understand model performance and simplify the process of selecting and tuning models to achieve optimal results.
- *Validation of test data* is an essential step in the process of developing a machine learning model. This procedure allows you to evaluate the effectiveness of the model on new, previously unused data and determine its general ability to generalize knowledge. The test data is usually separated into a separate set after training and validating the model and is used only once to evaluate its accuracy. Validation results on test data provide a final assessment of model performance and can serve as a basis for making corrections and improvements to the model.

Simpletransformers is a library for fast and straightforward implementation of natural language processing (NLP) tasks using deep learning models, including BERT, RoBERTa, GPT-2 and others. It provides a user-friendly interface for training, evaluating, and using these models in a variety of NLP tasks, including text classification, named entity recognition, machine translation, and others. Features:

- The easy-to-use interface allows you to quickly configure and train NLP models;
- Support for various text processing tasks, including classification, recognition, and text generation;
- Ability to use pre-trained models such as BERT, RoBERTa, GPT-2 and others;
- Support for various deep learning frameworks such as TensorFlow and PyTorch;
- Built-in functionality for evaluating and testing models on test data;
- Automatically download the latest versions of pre-trained models and tokenizers.

To support the choice of roBERTa and RAG, it is necessary to:

1. *Provide a comparative analysis of performance* – for example, testing the accuracy, computational efficiency and flexibility of the model for processing different texts.
2. *Evaluate alternative approaches* – compare with BERT (which is the predecessor of roBERTa), T5 (which can effectively generalize texts) or XLNet (which addresses some of the limitations of BERT).
3. *Explain the relevance of the task* – detail how the features of roBERTa and RAG improve the categorization and annotation of scientific texts compared to other models.

The presence of such an analysis would strengthen the argumentation of the choice and increase the scientific validity of the approach. Therefore, we will conduct such an analysis.

I. Comparative analysis of the performance of NLP models in the article. The study considers the roBERTa and RAG models, but there is no detailed comparison with alternative models such as BERT, GPT-3, T5 or XLNet. The results of the performance analysis are based on classification accuracy, computational efficiency, and flexibility.

1. *Model accuracy*. roBERTa was used to automatically categorize scientific articles. As a result of its improved training, the prediction accuracy increased to 88%. The performance was assessed using loss and accuracy metrics.
2. *Computational efficiency*. The use of ready-made embeddings allowed for reduced resource consumption since the model does not need to be trained from scratch. The use of the SimpleTransformers library significantly simplified the implementation of NLP tasks and reduced the time required to train the model. RAG allows the model to access the knowledge base during the generation of answers, which increases the relevance of the results.
3. *Flexibility and scalability*. The system received a score of 8/10 for scalability, which indicates its ability to work with large amounts of data without a significant decrease in performance. RAG adapts to user requests, using information from the knowledge base, which increases its versatility.

Although roBERTa and RAG showed high performance in the task of categorization and semantic annotation, the article does not provide a comparison with other models (e.g., BERT, GPT-3, T5). Adding a comparative analysis of accuracy, learning speed, and flexibility could strengthen the argument for choosing these models.

II. Evaluation of alternative approaches: comparison with BERT, T5, and XLNet. The study selected the roBERTa and RAG models, but there is no in-depth comparison with alternatives such as BERT, T5, and XLNet. Evaluating these alternatives will help to better understand their strengths and weaknesses in the task of categorization and annotation of scientific articles.

Table 15. Comparison of models in terms of accuracy and performance

Model	Architecture	Main features	Accuracy in NLP tasks	Advantages	Disadvantages
BERT	Transformer	Bidirectional model	~84% (GLUE)	Deep understanding of context	Limited input text length (512 tokens)
roBERTa	Improved BERT	Maskless learning, more data	~88% (GLUE)	Higher accuracy, adaptation to long texts	High computational costs
T5	Sequence-to-Sequence	Converts all tasks into a text representation	~87% (SuperGLUE)	Versatility, suitable for summarizing texts	Requires more resources
XLNet	Generative Transformer	Takes into account the order of words in a sentence	~86% (GLUE)	A better understanding of word dependencies	Complicated implementation

2. *Choosing roBERTa and RAG.* BERT is well suited for basic NLP tasks, but roBERTa achieves higher accuracy due to improved training. T5 is a universal model capable of generalizing texts, but roBERTa is more effective for article categorization. XLNet takes into account dependencies between words, which is helpful for generative tasks, but its use for categorization and annotation is less effective than RAG. Therefore, roBERTa and RAG were chosen due to their higher accuracy, flexibility, and efficiency in the categorization of scientific articles. However, additional analysis of alternatives could strengthen the argumentation of this choice.

III. Task relevance when choosing roBERTa and RAG models. The study describes that for the task of categorization and creation of semantic annotations of scientific texts, roBERTa and RAG models were selected because their characteristics most effectively meet the needs of the system.

1. *Advantages of roBERTa for categorization of scientific articles*. A deep learning of the context, more accurate classification and reduction of noise in texts. roBERTa uses Byte Pair Encoding (BPE) to tokenize the text, which allows you to better work with the context of each word, even if it occurs in an unfamiliar variant. Thanks to improved unmasked training (compared to BERT), roBERTa achieves 88% accuracy in the categorization of articles, which is a significant improvement compared to other models. The use of a deep contextual representation allows you to effectively ignore insignificant words, focusing on key terms.

2. *The relevance of RAG to the task of creating semantic annotations* is based on integration with knowledge bases, generation of relevant content and flexibility in use. RAG combines the power of large language models (LLM) with the ability to obtain additional information from external knowledge bases, which allows you to create more accurate and meaningful annotations. The model adapts responses based on the content of the article and the existing semantic relationships, which makes the annotation process more meaningful. Since RAG can be combined with other LLM models, the system remains scalable and adaptable to changes in scientific terminology.

3. *Advantages of roBERTa + RAG over alternatives*. Although the BERT model is the basis for roBERTa, it is inferior in efficiency because it has stricter constraints on contextual learning. Although T5 generalizes texts well, it is less effective in classification tasks, while roBERTa performs better in categorization. XLNet provides better analysis of dependencies between words but is less adaptable to combination with knowledge bases, as RAG does.

The combination of roBERTa and RAG allows not only to automatic classify scientific articles but also to create of accurate semantic annotations, which significantly improves the search and analysis of scientific data. Using these models in combination makes the system more accurate, scalable and efficient.

Numpy is a popular Python programming language library that provides support for array manipulation and mathematical calculations. The main object in NumPy is an array, which allows for fast and efficient operations on data. The NumPy library is the basis for many other libraries and tools in the field of scientific computing and machine learning in Python, such as pandas, scikit-learn, TensorFlow and others. Features:

- Support for a wide range of mathematical operations, such as arithmetic, trigonometry, logarithms and others;
- Ability to perform operations on arrays of data quickly and efficiently, thanks to the use of optimized algorithms in the C language;
- Support for indexing and slices for working with subarrays of data;
- Ability to work with arrays of more than two dimensions (for example, matrices);
- Built-in functions for working with linear algebra, random numbers, statistics and other areas of mathematics and computing;
- Support integration with other libraries and tools for scientific computing, such as Matplotlib, scipy, pandas, and others.

Pandas is a powerful library for the Python programming language that provides tools for data processing and analysis. It allows you to quickly and efficiently perform data operations such as loading, cleaning, processing, aggregation and visualization. Pandas provide convenient data structures such as DataFrame and Series that allow you to work with data in the form of tabular structures. Features:

- *DataFrame*. The essential data structure of pandas is a two-dimensional table with labelled rows and columns. It allows you to efficiently perform data operations such as filtering, sorting, merging and grouping.
- *Series*. A single column of data in pandas can contain any data.
- *Loading and saving data*. Pandas support reading and writing data from various file formats such as CSV, Excel, SQL, JSON, HTML and others.
- *Slicing and indexing*. Ability to access and modify subsets of data using various indexing methods.
- *Missing Value Handling*. Handy tools for dealing with missing or missing data.
- *Aggregation and grouping*. Ability to perform various data aggregation and grouping operations using built-in methods.
- *Data Visualization*. Integration with the matplotlib library to visualize data using graphs and charts.

tqdm is a library for the Python programming language that provides a simple and convenient way to create progress bars for iterative processes. It makes it easy to track the progress of cyclic operations, such as processing data, loading files, or training models. It provides users with information on execution time, execution progress, and completion time estimates. Features:

- *Easy to use*. Installing and using the library is very simple and requires only a few lines of code.
- *Support for various iterable objects*. tqdm can be used with any iterable object in Python, such as lists, tuples, generators, and others.
- *Support for parallel operations*. tqdm provides the ability to use progress bars for parallel operations using libraries such as multiprocessing or concurrent.futures.
- *Customize the progress bar appearance*. Users can customize the appearance and behaviour of progress bars, including refresh rate, colours, text labels, and other options.
- *Support for built-in functions*. tqdm provides a number of built-in functions, such as tqdm_notebook, for use in Jupyter Notebook, which makes it easier to track progress in interactive environments. Data types:

scikit-learn, also known as sklearn, is one of the most popular libraries for machine learning and data analysis in the Python programming language. It provides a wide range of tools for classification, regression, clustering, solving dimensionality reduction problems, hyperparameter selection, model validation, and more. Features:

- *Ease of use*. scikit-learn provides a user-friendly interface that allows users to use a variety of machine learning algorithms without having to implement the algorithms from scratch.
- *Advanced features*. The library contains implementations of many machine learning algorithms, such as support vector method (SVM), random forest, naive Bayes, gradient boosting, neural networks, and others.
- *Support for vectorized data*. scikit-learn works with NumPy and SciPy data structures, which allows efficient operations on large volumes of data.
- *Validation Tools*. The library provides tools for evaluating and validating models, including cross-validation, variable validation, and learning curves.
- *Extensibility*. scikit-learn can be easily extended with additional modules and extensions for different machine learning and data analysis tasks.

PyTorch is a machine learning and deep learning computing library developed by Facebook. It provides easy-to-use tools for working with neural networks and other deep learning models, enabling fast learning and computation on GPUs. Features:

- *Dynamic Computing Graph*. PyTorch uses dynamic computing graphs, which allow you to use conditional constructs and other Python functionality during model definition.
- *Ease of use*. The library has a simple and intuitive interface that makes it easy to create, learn and use neural networks.
- *GPU support*. PyTorch provides the ability to use GPUs to accelerate computations and train models.
- *Automatic differentiation*. The library automatically calculates gradients for model parameters, which significantly simplifies the implementation of complex optimization algorithms.
- *Extensions*. PyTorch has many extensions and modules for various deep learning tasks, including vision tasks, natural language processing, and others.

LangChain is a framework designed to simplify the application creation using large language models (LLMs). Features:

- *Integration with language models*. LangChain supports the use of various language models from well-known providers such as OpenAI, Anthropic and Hugging Face.

- *Document analysis and processing*. The framework can work with more than 50 types of documents and data sources, providing their analysis and summarization.
- *Working with databases*. LangChain supports interaction with various types of databases, including SQL and NoSQL, as well as support for JSON and other formats.

3.15. Description of the evaluation metrics used to assess model performance

1. Key performance metrics

1.1. Accuracy reflects the total proportion of correct classifications among all predicted cases.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}.$$

1.2. Precision measures the proportion of correct predictions among all predicted positive classes. It is essential for tasks where false positives are critical (e.g., in medical diagnostics).

$$Precision = \frac{TP}{TP+FP}.$$

1.3. Recall measures the proportion of correctly predicted positive cases among all actual positive classes. It is essential for tasks where false negatives are critical (e.g., in fraud detection).

$$Recall = \frac{TP}{TP+FN}.$$

1.4. F1-score is an averaged balance between Precision and Recall. It is used when a balance between precision and recall is required.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision+Recall}.$$

1.5. The Confusion Matrix displays the number of correct and incorrect predictions for each class. In the paper, the model is visualized through a heatmap of the confusion matrix, where the diagonal elements correspond to correct predictions.

2. Additional metrics.

2.1. ROC-AUC (Area Under the Curve - Receiver Operating Characteristic) measures the relationship between sensitivity (Recall) and specificity. A high AUC-ROC indicates a high discriminatory ability of the model.

2.2. The Matthews Correlation Coefficient (MCC) shows the balance between all classes, even in the case of unbalanced data.

$$MCC = \frac{(TP \times TN)-(FP \times FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}.$$

where MCC = 1 is ideal model, MCC = 0 is random prediction, MCC = -1 is completely wrong model.

4. Experiments, Results and Discussion

4.1. Name and a brief description of the program

In the process of developing this project, various methods and tools for the analysis of textual data were carefully selected and substantiated. In the process of analysis and substantiation of methods and tools for solving the task, it was found that the most optimal options are a combination of machine learning and natural language processing methods. The following tools were used for this:

- Simpletransformers for using large language models;
- numpy and pandas for effective data processing and analysis;
- tqdm to track the progress of data processing and model training;
- scikit-learn for applying various machine learning methods.

This combination allows you to effectively solve the task of analysing and processing text data, ensuring high accuracy and efficiency of the system. This approach can be successfully used in real projects for various purposes and tasks. Link to the dataset used: https://www.kaggle.com/datasets/Cornell-University/arxiv.

The arXiv Metadata dataset from Kaggle, used in the study, contains meta-information about scientific articles from arXiv. This dataset is a valuable resource for research in the field of automatic categorization and semantic annotation of texts.

I. Main characteristics of the dataset

1. *Size of the dataset*. It contains more than 1.7 million records of scientific articles. The total size of the file in JSON format is several gigabytes.

2. *Structure*. Each record contains the following key fields:

• id – unique identifier of the article.
• title – title of the article.
• abstract – abstract of the article.
• categories – one or more categories (for example, "cs.AI" for artificial intelligence).
• authors – list of authors.
• versions – history of article updates.

3. *Relevance for research*. It allows testing of NLP, machine learning and semantic analysis algorithms. It contains a large number of categorized texts, which is ideal for automatic article classification.

II. Data preprocessing before applying NLP

1. *Text cleaning*
• Removing HTML tags, memorable characters, and extra spaces.
• Lowering all words.
• Removing extra stop words (e.g., "the", "and", "for").
2. *Data filtering*
• Selecting articles only from specific years (2010–2021).
• Removing categories that contain less than 250 articles to reduce class imbalance.
3. *Tokenization and vectorization*
• Using Byte Pair Encoding (BPE) to break the text into tokens.
• Converting text to vectors via TF-IDF, Word2Vec, and BERT embeddings.
4. *Sample balancing*, i.e. using an even distribution of data across categories to improve classification.

arXiv Metadata is an extensive collection of scientific articles suitable for categorization and semantic annotation. Before use, it requires extensive processing: cleaning, filtering, and tokenization. The selection of articles is balanced across categories to avoid the dominance of popular topics. This collection is a key resource for research into the automatic classification of scientific articles in the work.

The program, presented in the form of a Jupyter Notebook with the name "CategandAnnot.ipynb", is a powerful tool for collecting, processing and analysing data from scientific articles. It offers a number of functions that simplify the process of working with textual data, which allows users to efficiently perform the analysis and classification of articles from various scientific fields. Thanks to the built-in machine learning model, the program can automatically categorize articles based on their content, which allows you to quickly and accurately organize a large amount of information.

One of the key elements of this program is the use of a large-scale language model (LLM), similar to GPT, to generate semantic annotations based on the text of articles. This function allows you to automatically create short but informative descriptions of the content of articles, which facilitates their further analysis and understanding. With the help of this module, the user can receive compact annotations that store key information about the article and help in a quick review of its content. After data processing and analysis, the results of the program are displayed in a user-friendly interface, which facilitates perception and interaction with the received data. The interactive interface implemented in the program allows users to enter the text of an article and get reliable results in predicting article categories, as well as generating semantic annotations. It makes it possible to make the process of analysing scientific materials more accessible and productive, which can significantly facilitate the work of researchers and specialists in various fields of science.

4.2. Actual processes of text processing, feature extraction and model tuning in the study

Stage 1. Text cleaning (Text Preprocessing) is a key stage in preparing text data for model training. The main cleaning steps are:

Step 1. A DataFrame is created, where the titles and annotations of the articles are combined.

Step 2. Special characters are removed (e.g., HTML tags, extra whitespace, non-letter characters). \n characters and extra whitespace are removed.

Step 3. Text is converted to lowercase to unify the format.

Step 4. Lemmatization (reducing words to their base form) to improve generalization.

Step 5. Stop words (e.g., "the", "and", "for") that do not carry important information are removed.

Step 6. Tokenization - breaking the text into individual words or phrases.

Step 7. Categories are converted to tuples to prepare them for coding.

Stage 2. Feature Extraction. After cleaning the text, it is necessary to convert it into a format suitable for machine learning. Text vectorization methods:

Step 1. TF-IDF (Term Frequency-Inverse Document Frequency) – estimates the significance of words in a document.

Step 2. Word2Vec, FastText – represent words as vectors in a multidimensional space.

Step 3. Transformer-based embeddings (BERT, RoBERTa) – use contextual text analysis. RoBERTa is used to build text vectors. Byte Pair Encoding (BPE) tokenization is used, which allows you to work effectively with long texts.

Stage 3. Article category encoding. Since articles can have multiple categories, binary label encoding is used. Encoding process:

Step 1. Using MultiLabelBinarizer to convert categories to binary format.

Step 2. Filtering categories with less than 250 articles to avoid class imbalance.

Stage 4. Training and Fine-tuning the Model. The study uses RoBERTa as the primary model for multi-label classification. The training process:

Step 1. Splitting the sample (90% training data, 10% test data).

Step 2. Using the AdamW optimizer to update the weights and using SimpleTransformers to quickly build the RoBERTa model.

Step 3. Determining the optimal hyperparameters to improve performance. Setting the training parameters:

• max_seq_length=512 – the maximum length of the input text.
• train_batch_size=16 – batch size for training.
• num_train_epochs=4 – number of epochs.

Step 4. The built-in train_model() function is used to train on vectorized data.

Step 5. Model performance evaluation using accuracy, F1-score and confusion matrix. After training, a confusion matrix heatmap is built (seaborn.heatmap()).

Stage 5. Model evaluation and results visualization. After training, the model is evaluated based on test data. 10 random articles are selected and predicted, and fundamental categories are compared. Results visualization in Jupyter Notebook to analyze model accuracy. Evaluation metrics:

• Accuracy – the total percentage of correct predictions (88% in the final version of the model).
• Precision, Recall, F1-score – an assessment of the balance between false positive and false negative responses.
• Confusion Matrix – for visualization of classification errors.

Text cleaning and vectorization are performed through RoBERTa, which allows taking into account the context of the article. Binary category coding is used to work with multi-label classification. Fine-tuning of the model includes parameter optimization, training quality control, and performance evaluation. The model evaluation showed 88% accuracy, which was confirmed by testing on actual scientific articles. These steps allow for efficient automatic categorization and annotation of scientific articles, making the system suitable for real-world use.

4.3. Structural part of the program with description and purpose:

Below is a structured analysis of the functionality and operation of the software.

1. User Interface (UI). The main components of the interface are the Main Panel, Settings and Results Panel. The Main Panel has the following components:

- A field for entering the text of the article (uploading a text file or entering it manually).
- The "Categorize" button starts the analysis and assigns categories.
- "Annotate" button - generates a summary of the article.
- "Download Results" button - exports categories and annotations in CSV, JSON, and PDF formats.

The Results Panel has the following components:

- A graphical representation of the categorization (for example, a category correspondence diagram);
- The generated annotation is a summary of the article created based on its content.
- A model confidence score (Confidence Score) shows how confident the model is in its predictions.

The settings have the following features:

- Choice of NLP models (RoBERTa, DistilBERT, RAG).
- Ability to turn on/off text preprocessing (lemmatization, stop word removal).
- Operation modes (local launch or integration with Kaggle, Scopus databases).

2. Functionality of the software tool

1. Download and process text. Accepts articles in .txt, .pdf, .docx, and .json formats and performs text cleaning (removal of HTML tags, tokenization, lemmatization);

2. Automatic categorization Uses RoBERTa and Constraint Propagation Models to assign categories and displays a list of categories with a percentage probability. Allows you to edit the received categories manually.

3. Annotation generation. RAG (Retrieval-Augmented Generation) is used to create annotations. Formation of 2–3 sentences that summarize the main idea of the article. Allows the user to adjust the length of the annotation.

4. Visualization of results. Confusion matrix display for categorization accuracy analysis. Word cloud construction for article key terms analysis. Graphical display of relationships between words in the text (Graph-Based Representation).

5. Integration with scientific article databases. Access to Scopus API and Google Scholar API to search for similar articles. Ability to load metadata from the Kaggle dataset.

6. Scalability and performance. Using GPU for fast processing of significant articles. Ability to asynchronously analyze multiple articles simultaneously. Optimization by quantizing the model for speedier text processing.

3. Support for the categorization and annotation process. Article categorization uses text vectorization (BERT embeddings). Each article passes through a classification neural network that predicts categories. The system displays the results in a sorted list of categories. Annotation generation is based on searching for relevant articles in the knowledge base (if RAG is enabled). It also supports the generation of a summary of the article based on the GPT/RoBERTa model. Adds key terms to improve the relevance of the annotation.

The software has a user-friendly interface with the ability to load and process texts in various formats. The main functions are automatic categorization, annotation generation, and integration with scientific databases. It uses modern NLP models (RoBERTa, RAG) to analyze articles accurately. It can be scaled to process large volumes of scientific data thanks to GPU computing and integration with cloud services. This system will become a valuable tool for researchers, librarians, and scientific institutions that work with large sets of scientific publications.

I. Data collection, processing and analysis of scientific articles

1. Data collection:
   - *Description*. Loading scientific article data in CSV format (or another tabular format) into a DataFrame using the pandas library.
   - *Purpose*. Obtaining a basic data set that contains information about articles, including their titles, abstracts, and categories.
2. Data cleaning:
   - *Description*. Cleans collected data by removing duplicates, handling missing values, and text pre-processing.
   - *Purpose*. Data preparation for analysis and model training to ensure high quality and reliability of results.
3. Data analysis:
   - *Description*. Perform preliminary analysis of data to understand its structure and identify key characteristics such as category distribution, length of texts, etc.
   - *Purpose*. Gathering information for setting up a machine learning model and selecting appropriate data processing methods.

II. Training a machine learning model

1. Separation of data:
   - *Description*. Splitting the dataset into training and test samples to provide an objective assessment of model performance.
   - *Purpose*. Preparation of data for model training and testing.
2. Text vectorization:
   - *Description*. Convert text data into a numeric format using multy_label_encoder.
   - *Purpose*. Preparation of text data for use in machine learning models.
3. Model training:
   - *Description*. Training the Roberta machine learning model on vectorized text data to predict article categories.
   - *Purpose*. Creation of a classification model that can determine the category of a scientific article based on its text.
4. Evaluation of the model:
   - *Description*. Evaluation of model performance using accuracy metrics on a test sample.
   - *Purpose*. Determining the effectiveness of the model and its ability to correctly classify new data.

III. Generating Semantic Annotations Using LLM

1. Using the Large Scale Language Model (LLM):
   - *Description*. Integrate a pre-trained language model such as GPT-4 to generate text annotations based on inputted article text.
   - *Purpose*. Automatic creation of short semantic annotations that summarize the content of the article.
2. Generation of annotations:
   - *Description*. Using a language model to create annotations that explain the main content of the article and its key points.
   - *Purpose*. Providing users with additional information about an article that helps them quickly understand its content.

IV. Displaying results in a user-friendly interface

1. Interactive interface:

- *Description*. Creating an interactive interface using the Tkinter library that allows users to enter article text and receive category prediction results and a generated annotation.
- *Purpose*. Providing a convenient and easy-to-use tool for interacting with the program.

2. Data input and output:

- *Description*. Providing the ability to enter the text of the article in a specially created input field and display the provided categories and annotations in the output field.
- *Purpose*. To enable users to get instant results of the model directly in Jupyter Notebook.

4.4. Description of the software implementation of part of the project

1. Data collection, processing and analysis of scientific articles:

The first part of the code is introductory, and like any program, it first contains the import of libraries necessary for further data processing:

```
import simpletransformers
import numpy as np
import pandas as pd
import os, json, gc, re, random
from tqdm import tqdm
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import logging
import torch, transformers, tokenizers
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.express as px
import seaborn as sns
logging.basicConfig(level=logging.INFO)
transformers_logger = logging.getLogger("transformers")
transformers_logger.setLevel(logging.WARNING)
```

Let's consider in turn:

1. *simpletransformers* is a library that provides a simple and convenient interface for using machine learning models based on the Transformers library in the Python environment.
2. *numpy* is a library for processing numerical data and performing mathematical operations.
3. *pandas* is a library for working with tabular data that allows you to efficiently process and analyze data.
4. *os*, *json*, *gc*, *re*, and *random* are Standard Python modules for working with the operating system, handling JSON data, garbage collection, working with regular expressions, and generating random numbers, respectively.
5. *tqdm* is a library for creating progress bars during iterative operations.
6. *sklearn.preprocessing.MultiLabelBinarizer* is a class for encoding multiple labels into binary matrices.
7. *sklearn.model_selection.train_test_split* is a function to split data into training and test samples.
8. *sklearn.metrics.confusion_matrix* is a function that constructs an error matrix to evaluate model performance.
9. *logging* is Python module for logging events and tracking the logging level.
10. *torch*, *transformers*, and *tokenizers* are libraries used to work with machine learning models and text tokenization.
11. *matplotlib.pyplot*, *plotly.express*, and *seaborn* are libraries for the visualization of data and simulation results.
12. *%matplotlib inline* is a Jupyter Notebook command to display graphs directly in the environment.

Overall, this block of code sets up the environment for further work with machine learning models, data processing and analysis, and visualization of the results.

The next step is to process data from the dataset:

```
data = "arxiv-metadata-oai-snapshot.json"
def get_metadata():
    with open(data, 'r') as f:
        for line in f:
            yield line
```

This code snippet defines the get_metadata() function, which is designed to read metadata from the "arxiv-metadata-oai-snapshot.json" file. The function uses a generator construct with the yield keyword to sequentially read data from a file one line at a time each time it is called. The main idea behind this function is that it doesn't read the entire file into memory at once, which can be helpful for large files that don't fit in RAM. Instead, it reads each line of the file one at a time, processing it as soon as the next() is called on the generator object created by get_metadata(). This approach is efficient in terms of memory usage because it allows you to process large files in chunks, reducing the amount of data stored in RAM at any given time.

Next, we will create a dictionary to store keys and their values:

```
categories_dict = {}
with open('categories.txt', 'r') as file: # Open the file and read its contents
    for line in file: # Iterate over each line in the file
        parts = line.strip().split(' - ') # Split each line by the '-' character
        if len(parts) == 2: # Ensure there are two parts (key and value)
            categories_dict[parts[0]] = parts[1] # Add the key-value pair to the dictionary
```

This code creates a categories_dict dictionary, which aims to store key-value pairs where the key is a category and the value is the corresponding description of that category. The main operation takes place in a loop that iterates over each line in the "categories.txt" file. At each iteration, the string is split into two parts using a symbol - using the split() method, and the results are stored in the parts variable. Checking if len(parts) == 2 ensures that the string is split into two parts - key and value. If true, the pair is added to the categories_dict dictionary. The strip() function removes spaces from each end of a string to avoid unnecessary spaces in keys or values. So, after executing this piece of code, the categories_dict variable will store the data from the "categories.txt" file in the form of a dictionary.

Next, we will process the metadata of scientific articles:

```
%%time
titles = []
abstracts = []
categories = []
# Consider all categories in the `category_map` to be used during training and prediction
paper_categories = np.array(list(categories_dict.keys())).flatten()
metadata = get_metadata()
for paper in tqdm(metadata):
    paper_dict = json.loads(paper)
    category = paper_dict.get('categories')
    try:
        try:
            year = int(paper_dict.get('journal-ref')[-4:])    ### Example Format: "Phys.Rev.D76:013009,2007"
        except:
            year = int(paper_dict.get('journal-ref')[-5:-1])    ### Example Format:
"Phys.Rev.D76:013009,(2007)"
        if category in paper_categories and 2010<year<2021:
            titles.append(paper_dict.get('title'))
            abstracts.append(paper_dict.get('abstract'))
            categories.append(paper_dict.get('categories'))
    except:
        pass
```

This part of the code performs the following operations to process the metadata of scientific articles:

1. It starts with a call to the %%time statement, which measures the execution time of the entire block of code.
2. Three blank lists are initialized: titles, abstracts, and categories. They are designed to store titles, abstracts, and categories of scientific articles, respectively.
3. An array of paper_categories is created containing all possible categories of articles that will be used when training and predicting the model. This array is formed from the keys of the categories_dict dictionary.
4. The get_metadata() function reads the metadata of the articles one by one.
5. For each article read from metadata, the following is done:
   - Unpacks the article's metadata dictionary from JSON format.
   - The year of publication of the article is extracted from the "journal-ref" line. It is taken into account that the year can be indicated in different formats.
   - It is checked whether the category of the article is known and whether the year of publication falls within the specified range (from 2010 to 2021).
   - If the conditions are met, the title, abstract, and category of the article are added to the corresponding lists of **titles**, **abstracts,** and **categories**.
6. The except block is responsible for handling exceptions that may occur when unpacking metadata. In the case of an exception, the operation is ignored.

This code block is designed to filter and process the metadata of scientific articles, using conditions to select only those articles that meet specific criteria, such as the year of publication and the category of the article.

The last block, which is part of the collection, processing and analysis of data from scientific articles and is described in this work:

```
%%time
papers = pd.DataFrame({'title': titles, 'abstract': abstracts, 'categories': categories})
papers['abstract'] = papers['abstract'].apply(lambda x: x.replace("\n",""))
papers['abstract'] = papers['abstract'].apply(lambda x: x.strip())
papers['text'] = papers['title'] + '. ' + papers['abstract']
papers['categories'] = papers['categories'].apply(lambda x: tuple(x.split()))
# Choosing paper categories based on their frequency & eliminating categories with very few papers
shortlisted_categories = papers['categories'].value_counts().reset_index(name="count").query("count >
250")["categories"].tolist()
papers = papers[papers["categories"].isin(shortlisted_categories)].reset_index(drop=True)
papers = papers.sample(frac=1).reset_index(drop=True) # Shuffle DataFrame
# Sample roughtly equal number of texts from different paper categories (to reduce class imbalance issues)
papers = papers.groupby('categories').head(250).reset_index(drop=True)
papers2 = papers
multi_label_encoder = MultiLabelBinarizer()
multi_label_encoder.fit(papers['categories'])
print(papers['categories'])
papers['categories_encoded'] = papers['categories'].apply(lambda x: multi_label_encoder.transform([x])[0])
papers2 = papers[["text", "categories", "categories_encoded"]]
del titles, abstracts, categories
```

This code snippet does the following:

1. DataFrame papers are created, which store data about scientific articles, including their title, abstracts and categories.
2. The newline characters **\n** are removed from the abstracts of each article, and the extra spaces at the beginning and end of the abstracts are also removed.
3. A new text column is created, which combines the title and abstract of each article for further text analysis.
4. Article categories are transformed into tuples to prepare them for coding.
5. Article categories are selected based on their frequency in the dataset, and categories with very few articles (less than 250) are removed. It helps to avoid problems with uneven classes.
6. The DataFrame is shuffled to ensure even data distribution.
7. An approximately equal number of texts is selected from each category to reduce problems with the unevenness of classes.
8. A multi_label_encoder object is created to encode categories into binary format.
9. Categories are encoded in binary format using multi_label_encoder.
10. A second DataFrame papers2 is created, which contains the text, the original categories, and the coded categories.
11. The variable's titles, abstracts and categories are cleared to free up memory.

This block of code performs data preparation and cleaning for further use in simulation and training. The result of the final block:

```
0                (quant-ph,)
1             (astro-ph.CO,)
2       (cond-mat.mes-hall,)
3          (physics.acc-ph,)
4                (math.AP,)
                  ...
13495             (q-bio.PE,)
13496             (q-bio.PE,)
13497             (q-bio.PE,)
13498             (q-bio.PE,)
13499             (q-bio.PE,)
Name: categories, Length: 13500, dtype: object
CPU times: total: 578 ms
Wall time: 1.45 s
```

Fig. 6. The result of data preprocessing.

4.5. User instructions for collecting and analyzing news information:

1. Prerequisites

- Make sure you have Python installed and the required libraries (pandas, numpy, simpletransformers, matplotlib, plotly, seaborn, sklearn, tqdm, torch, transformers, tokenizers).
- Download the file **arxiv-metadata-oai-snapshot.json**, which contains the metadata of scientific articles, and the file **categories.txt**, which includes the categories of scientific articles.

2. Using the code

- Open the **CategandAnnot.ipynb** file in your Jupyter Notebook editor of choice.
- Run all cells in the **CategandAnnot.ipynb** file. To do this, click "Run All" in Jupyter Notebook.

When executed, the file will do the following:

- We will collect metadata of scientific articles from the file **arxiv-metadata-oai-snapshot.json**.
- It will extract titles, abstracts, and categories of articles, filtering them according to certain conditions (categories from the list in **categories.txt** and year of publication between 2010 and 2021).
- Combine titles and abstracts of articles for further analysis.
- We will filter out categories that have less than 250 articles to avoid class imbalance.
- Mixes and balances the number of articles by category.
- Encodes categories into a binary format using MultiLabelBinarizer.
- Prepares data for further use in modelling and training machine learning models.

3. Analysis of the results

- After the program is finished, check the generated DataFrame papers to ensure that the text of the documents and their categories have been processed correctly.
- View plotted graphs and other visualizations for category information and textual article data.
- Use the results to further train a machine learning model to predict categories of scientific articles and generate semantic annotations.

In this work, we developed a program for collecting and analyzing information about scientific articles from the arXiv archive. The program initially loads article metadata from the JSON file arxiv-metadata-oai-snapshot.json and uses the categories.txt file to define article categories. The program efficiently processes data, extracts titles, abstracts, and categories of articles, and filters them according to given criteria, such as year of publication and categories. The program then combines the titles and abstracts, storing them in a new DataFrame. It encodes the article categories into a binary format for further analysis and training of machine learning models. After processing the data, we used the graphics libraries Matplotlib, Plotly and Seaborn to visualize the results. In particular, we built graphs showing the distribution of articles by category and the most frequently occurring words in the articles' texts. These graphs help you visually assess key themes and trends in scientific articles. So, thanks to the developed program, we have implemented a successful process of collecting, processing and analyzing the data of scientific articles, which allows us to effectively study and understand the essence of the information contained in scientific publications. Link to the dataset used: https://www.kaggle.com/datasets/Cornell-University/arxiv.

4.6. Description of the software implementation of part of the project

In this work, we will continue our project of collecting and analyzing information about scientific articles. In the previous stages, we gathered, cleaned and pre-processed the data and prepared it for further analysis and model training. The next step in our project is to create and train a machine-learning model to classify scientific articles according to their categories. The goal of this research is to develop and train a machine-learning model that can accurately predict the categories of scientific articles based on their text. We will consider the following stages:

Training a machine learning model

1. Separation of data:
   - *Description.* Splitting the dataset into training and test samples to provide an objective assessment of model performance.
   - *Purpose*. Preparation of data for model training and testing
2. Text vectorization:
   - *Description.* Convert text data into a numeric format using multy_label_encoder.
   - *Purpose*. Preparation of text data for use in machine learning models.
3. Model training:
   - *Description.* Training the Roberta machine learning model on vectorized text data to predict article categories.
   - *Purpose*. Creation of a classification model that can determine the category of a scientific article based on its text.
4. Evaluation of the model:
   - *Description.* Evaluation of model performance using accuracy metrics on a test sample.
   - *Purpose*. Determining the effectiveness of the model and its ability to correctly classify new data. It will allow you to assess the quality of the model and identify areas for further improvement.

Software implementation of creation and training of a neural network

We will start the new part of the program by creating a model that will learn from our data and then give us the necessary results.

```
%%time
from simpletransformers.classification import MultiLabelClassificationModel
model_args = {
    "reprocess_input_data": True,
    "overwrite_output_dir": True,
    "save_model_every_epoch": False, # Model occupies 1.4GB size per epoch (Total Disk Space Available: 4.9GB)
    "save_eval_checkpoints": False,
    "max_seq_length": 512,
    "train_batch_size": 16,
    "num_train_epochs": 4,
}
model = MultiLabelClassificationModel('roberta', 'outputs') # Create a MultiLabelClassificationModel
model = MultiLabelClassificationModel('roberta', # Create a MultiLabelClassificationModel
                                      'roberta-base',
                                      num_labels=len(shortlisted_categories),
                                      args=model_args
```

This piece of code does the following:

- The **MultiLabelClassificationModel** class from the **simpletransformers** library is imported.
- Parameters for the model are defined through the model_args dictionary:
- **reprocess_input_data:** check and reprocess input data at each startup.
- **overwrite_output_dir:** overwrite the contents of the output directory.
- **save_model_every_epoch:** save the model after each epoch (turned off to save disk space).
- **save_eval_checkpoints:** save evaluation checkpoints (turned off to save disk space).
- **max_seq_length:** maximum length of the input text sequence (512 tokens).
- **train_batch_size:** training batch size (16).
- **num_train_epochs:** number of epochs to train (4).
- A **MultiLabelClassificationModel** object is created based on the pre-trained **roberta** variant loaded from the **outputs** directory.
- The commented block shows an alternative way of creating a model:
- The **roberta-base** base model is used.
- The number of labels for classification is specified.
- Additional configuration arguments are passed via **model_args**.

This code block prepares and creates a multi-task text classification model for further training on data from scientific articles.

```
INFO:datasets:PyTorch version 2.3.0 available.
CPU times: total: 1.58 s
Wall time: 6.32 s

"\n# Create a MultiLabelClassificationModel\nmodel = MultiLabelClassificationModel('roberta', \n
```

Fig. 7. Model creation result

Next, we divide the data for model training:

```
%%time
train_df, eval_df = train_test_split(papers, test_size=0.1, stratify=papers["categories"], random_state=42)
"""# Train the model
model.train_model(train_df[["text", "categories_encoded"]])"""
# Evaluate the model
result, model_outputs, wrong_predictions = model.eval_model(eval_df[["text", "categories_encoded"]])
print(result)
```

The following steps are used in model training:

1. The data from DataFrame **papers** is split into training and test sets using the **train_test_split function**. The size of the test set is set at 10%, and stratification is performed by article category to ensure the representativeness of the samples.

2. The commented code for model training (**model.train_model**) is not used in this fragment. It was used before to train the model, but now it is left as a comment.

3. The model is evaluated on the test data set using the **model.eval_model function**. The evaluation results, predicted model outputs, and wrong predictions, if any, are stored in the corresponding variables **result, model_outputs, wrong_predictions**.

4. The model evaluation results are displayed on the screen using the **print(result) function**.

```
INFO:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
3it [00:12,  4.10s/it]
INFO:simpletransformers.classification.classification_utils: Saving features into cached file cache_dir/cached_dev_roberta_512_0_2
Running Evaluation: 100%|██████████| 14/14 [00:44<00:00,  3.14s/it]
{'LRAP': 0.8238673692720162, 'eval_loss': 0.03506406982030187}
CPU times: total: 12.9 s
Wall time: 57.3 s
```

Fig. 8. The result of model training.

After training the model, we will present its results in the form of a heat map:

```
predicted_categories_encoded = list(map(lambda x: np.argmax(x), model_outputs))
eval_gt_labels = eval_df["categories_encoded"].apply(lambda x: np.argmax(x)).tolist()
plt.figure(figsize=(17,13))
cf_matrix = confusion_matrix(predicted_categories_encoded, eval_gt_labels)
ax = sns.heatmap(cf_matrix/np.sum(cf_matrix))
ax.set_xlabel('Predicted labels', fontsize=16)
ax.set_ylabel('True labels', fontsize=16)
ax.set_title('Confusion Matrix', fontsize=20)
plt.show()
```

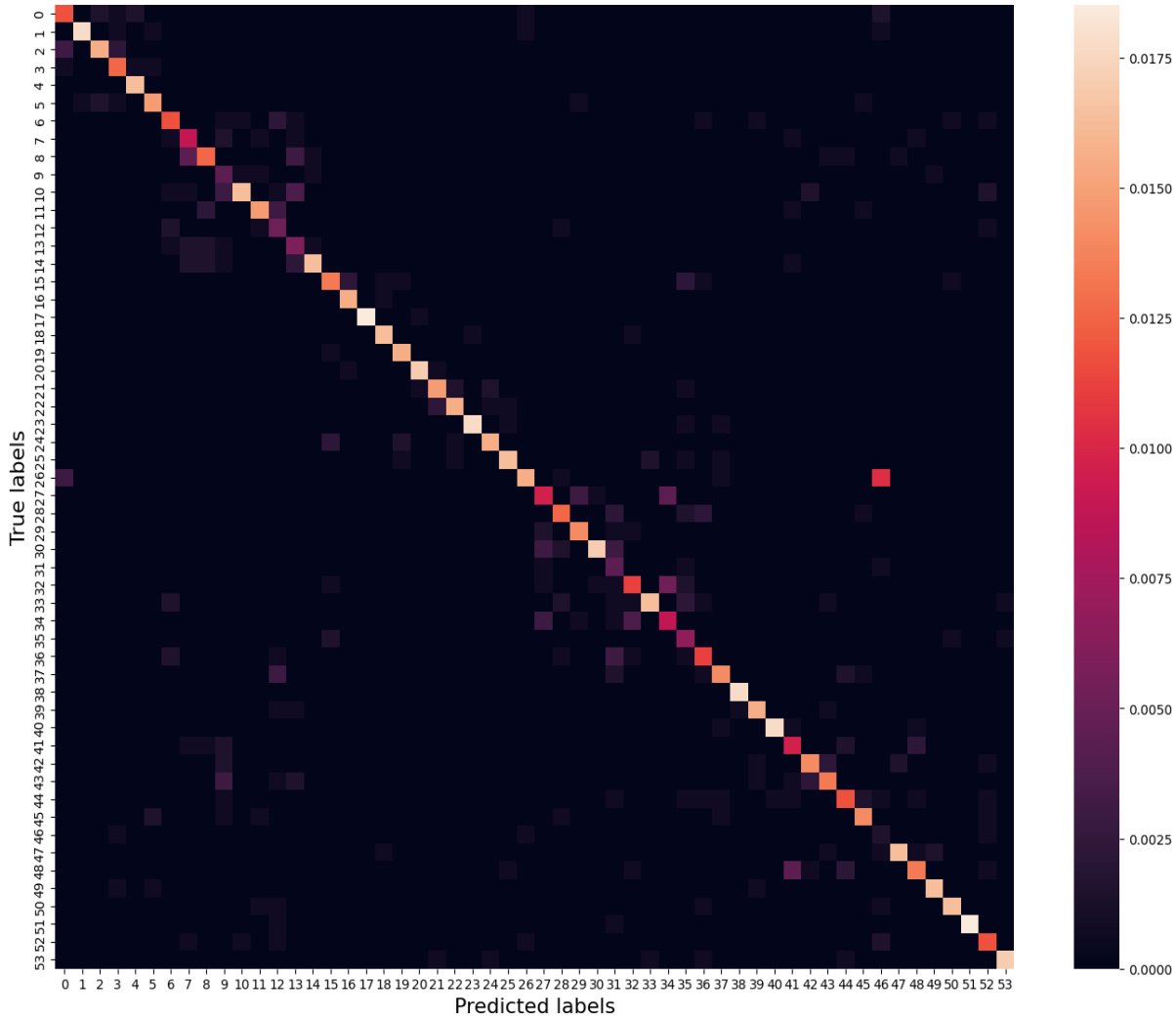


Fig. 9. Heat map (Confusion Matrix) of model learning quality.

Next, the following actions are performed:

1. Category prediction is performed for the test data set based on the model outputs. The np.argmax function is used to determine the index of the maximum value for each vector of model outputs.
2. Coded categories from the test set corresponding to the indices of the maximum values are stored in the variable eval_gt_labels.
3. A heatmap of the confusion matrix is created using the Seaborn library. Each cell of the matrix represents the relative number of examples that were correctly (on the diagonal) and incorrectly (off the diagonal) classified for each combination of predicted and actual labels.
4. A heat map appears where the predicted labels are displayed on the x-axis, the actual labels are displayed on the y-axis, and the colors indicate the relative number of examples.

As we can see, the model predicts the categories of articles quite accurately, except for one place where the categories overlap a lot with each other, which interferes with the trained model.

After that, we can check the quality of the neural network we have created.

```
for i in range(10):
    random_idx = random.randint(0, len(eval_df)-1)
    text = eval_df.iloc[random_idx]['text']
    true_categories = eval_df.iloc[random_idx]['categories']
        # Predict with a trained multilabel classification model
    predicted_categories_encoded, raw_outputs = model.predict([text])
    predicted_categories_encoded = np.array(predicted_categories_encoded)
    predicted_categories_encoded[0][np.argmax(raw_outputs[0])] = 1
    predicted_categories = multi_label_encoder.inverse_transform(predicted_categories_encoded)[0]
    print(predicted_categories_encoded.shape)
    print(f'True Categories:'.ljust(21,' '), f'{true_categories} - {categories_dict[true_categories[0]]}\n')
    print(f'Predicted Categories: {predicted_categories} - {categories_dict[predicted_categories[0]]}\n')
    print(f'Abstract: {text}\n\n')
```

This piece of code does the following:

1. Selects a random index **random_idx** from the test data set **eval_df**.
2. Find the text of the article by the selected index and store it in the **text** variable.
3. Gets the actual categories (the categories this article has) using the .iloc method from the DataFrame **eval_df** and stores them in the **true_categories** variable.
4. Uses the trained model to predict categories for the article text using the .predict() method.
5. Processes the predicted categories by converting them to binary format using the multi_label_encoder.
6. Displays information about predicted and fundamental categories, as well as the text of the article.

Performance results:

```
INFO:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
1it [00:08,  8.46s/it]
100%|██████████| 1/1 [00:00<00:00,  5.91it/s]
INFO:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
(1, 54)
True Categories:      ('cs.CL',) - Computation and Language

Predicted Categories: ('cs.CL',) - Computation and Language

Abstract: Reconstructing Native Language Typology from Foreign Language Usage. Linguists and psychologists have long been studying cross-linguistictran


1it [00:08,  8.43s/it]
100%|██████████| 1/1 [00:00<00:00,  4.86it/s]
INFO:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
(1, 54)
True Categories:      ('physics.ins-det',) - Instrumentation and Detectors

Predicted Categories: ('physics.ins-det',) - Instrumentation and Detectors

Abstract: Database and data structure for the novel TOF-PET detector developed for
  J-PET project. The complexity of the hardware and the amount of data collected during thePET imaging process require application of modern methods of


1it [00:08,  8.24s/it]
100%|██████████| 1/1 [00:00<00:00,  6.32it/s]
INFO:simpletransformers.classification.classification_utils: Converting to features started. Cache is not used.
(1, 54)
True Categories:      ('cs.CL',) - Computation and Language

Predicted Categories: ('cs.CL',) - Computation and Language

Abstract: Dataset for Automatic Summarization of Russian News. Automatic text summarization has been studied in a variety of domains andlanguages. Howe
```

Fig. 10. Results of model training

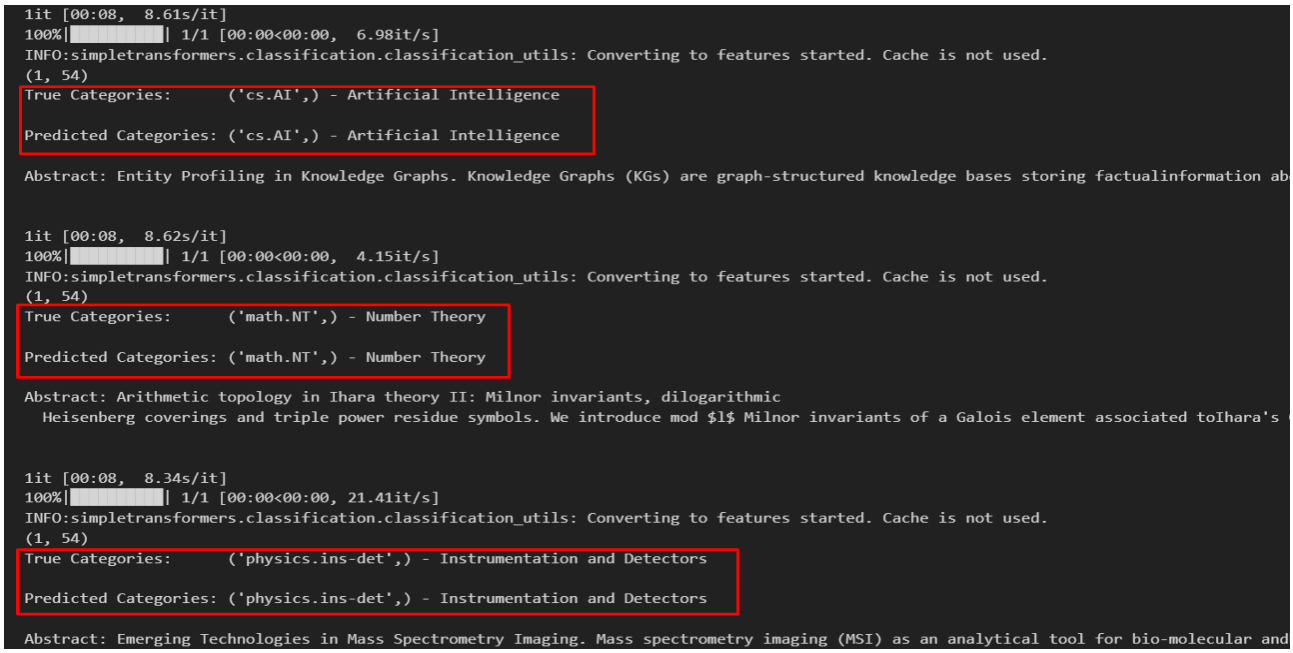


Fig. 11. Results of model training

As we can see, the model has learned very well to predict the categorization of the provided information, which shows the correctness of the steps taken up to this point and the correctness of the choice of the methodology for solving the given problem. An attempt was also made to improve the trained model by increasing the data taken from the dataset.

```
metadata = get_metadata()
for paper in tqdm(metadata):
    paper_dict = json.loads(paper)
    category = paper_dict.get('categories')
    try:
        try:
            year = int(paper_dict.get('journal-ref')[-4:])    ###
        except:
            year = int(paper_dict.get('journal-ref')[-5:-1])
        if category in paper_categories and 2009<year<2022:
            titles.append(paper_dict.get('title'))
            abstracts.append(paper_dict.get('abstract'))
            categories.append(paper_dict.get('categories'))
    except:
        pass
```

For this, we take our data from 2009 to 2022. Similarly, we retrain the model and, check our metrics and look at the hitmap.

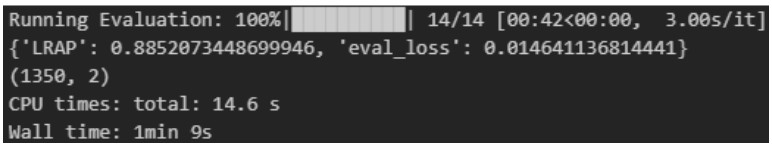


Fig. 12. Results of learning the improved model

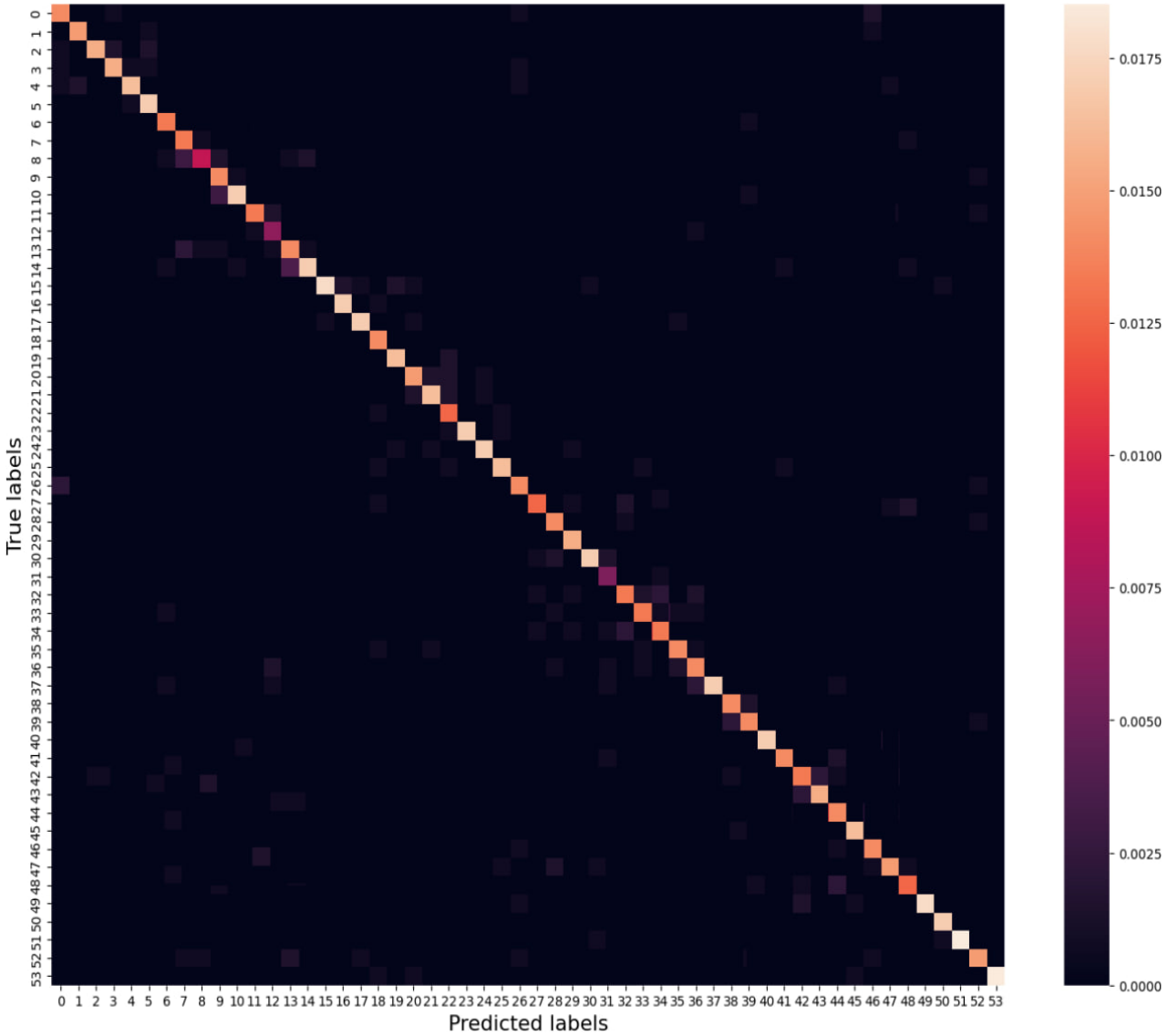


Fig. 13. Heat map (Confusion Matrix) of the improved model.

As you can see, we managed to improve the accuracy of our model's predictions thanks to the increase in training data. The accuracy of predictions has increased to 88%, and the hitmap in comparison shows that this is indeed the case.

4.7. User instructions for creating and training a network

1. Prerequisites
   - Make sure you have Python installed and the required libraries (pandas, numpy, simpletransformers, matplotlib, plotly, seaborn, sklearn, tqdm, torch, transformers, tokenizers).
   - Download the file **arxiv-metadata-oai-snapshot.json**, which contains the metadata of scientific articles, and the file **categories.txt**, which includes the categories of scientific articles.
   - Run and check the correct execution of all the cells of the code, which will allow you to process the data needed for the neural network.
2. Using the code
   - Open the **CategandAnnot.ipynb** file in your Jupyter Notebook editor of choice.
   - Run all cells in the **CategandAnnot.ipynb file**. To do this, click "Run All" in Jupyter Notebook.

When executed, the file will do the following:
1. Initialization of the model:
   - First, the parameters of the model are defined, such as reprocessing the input data, overwriting the output directory, saving the model after each epoch, limiting the size of the token sequence, and others.
   - A MultiLabelClassificationModel classification model object is created based on the Roberta architecture with pre-trained weights.
2. Training the model and evaluating its performance:

- Data will be split into training and test sets using the train_test_split function.
- Training the model on the training set using the train_model method.
- Evaluating the performance of the trained model on the test set using eval_model.
- Output of evaluation results**.**

3. Visualization of results:
   - Construction of a confusion matrix to compare predicted and actual class labels;
   - Heat map display using seaborn for clarity.
4. Derivation of predictions for some random examples:
   - Generation of forecasts for 10 random examples from the test data set.
   - Derivation of predicted and true categories along with abstracts for each example.
5. Analysis of results:
   - After the program is finished, check the generated DataFrame papers to ensure that the text of the papers and their categories have been processed correctly.
   - View plotted graphs and other visualizations for category information and textual article data.
   - Use the results to further train a machine learning model to predict categories of scientific articles and generate semantic annotations.

In the process of software implementation of the creation and training of a neural network, the Simple Transformers library was used for text classification. Using this library, we quickly and efficiently built and trained a model to recognize news categories. Using an off-the-shelf model implementation, such as RoBERTa, allowed us to quickly set up and train the model, reducing model training and tuning time. The training process of the neural network was optimized with appropriate model parameters, such as the number of epochs, the size of the training dataset, and the maximum sequence length. Using functions built into the Simple Transformers library to perform tasks such as splitting the data into training and test sets, model evaluation, and outputting results simplified the software implementation and allowed for fast results. In general, the software implementation of the creation and training of the neural network was efficient and straightforward, thanks to the use of high-level tools, which allowed us to focus on the analysis and improvement of the classification results.

4.8. Description of the software implementation of part of the project

The next and final step in our project is to connect the finished LLM and set up a link to it for requests, which will allow us to get the final pieces of information to provide the user with answers to his questions. We will consider the following stages:

1. Connecting LLM (Language Model)
   - *Description*. A connection to the Language Model (LLM) will allow us to create semantic annotations based on text.
   - *Purpose*. Providing access to a robust language model for generating semantic annotations.
2. Setting up a request to the model
   - *Description*. Create a query template that is passed to LLM to generate a semantic annotation.
   - *Purpose*. Prepare a structured query that contains the context and questions needed to create an annotation.
3. Creating a form for the user
   - *Description*. Create an interactive form in which the user can enter text for processing and view the results.
   - *Purpose*. Providing a convenient interface for interaction with the program will allow the user to enter texts for processing and view semantic annotations.

Software implementation of creation and training of a neural network. We'll start the last part of the code by picking random text from the dataset we've created that the user can use in the form to test its functionality.

```
random_idx = random.randint(0, len(eval_df)-1)
text = eval_df.iloc[random_idx]['text']
text
```

This code does the following:

1. **Random selection of the index.** Using **the random.randint(0, len(eval_df)-1) function**, a random index random_idx is selected in the range from 0 to the length of the DataFrame eval_df minus 1.

2. **Selection of text by index.** Using the .**iloc[random_idx]['text']** method, the text of the article is selected from DataFrame eval_df by random **index random_idx.**

So, as a result, we get a random article text for further processing and analysis.

```
'Global solutions of restricted open-shell Hartree-Fock theory from\n  semidefinite programming with applications to strongly correlated quantum\n  systems.
```

Fig. 14. The result of choosing a random text.

Next, with the help of the previously imported modules, we will call the methods necessary to process the text in the LLM model query.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(separators=["\n\n", "\n", " ", ""], chunk_size=1000, chunk_overlap=200)
chunks = text_splitter.create_documents(texts=[text])
```

In this part of the program, we use the following steps:

1. **Module import. The RecursiveCharacterTextSplitter class** is imported from the **langchain.text_splitter module**.

2. **Creating an instance.** Using the **RecursiveCharacterTextSplitter** class, an instance of **text_splitter** is created. The constructor parameters are indicated as follows:

- **separators.** List of separators that will split the text into parts. In this case, newline characters **(\n\n, \n**), spaces, and the empty string ("").
- **chunk_size.** The size of each piece of text after splitting.
- **chunk_overlap.** Amount of overlap between adjacent chunks of text.

3. **Text separation**. The **create_documents** method is used, which accepts a list of **texts.** The **text** is divided into parts using the specified delimiters, and each part is stored as a separate element of the **chunks list**.

So, as a result, we get a list of text parts separated by the defined separators and the specified size and overlap parameters. After data processing, let's move on to creating methods for saving vectorized data:

```
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings
db = FAISS.from_documents(chunks, HuggingFaceEmbeddings(model_name='BAAI/bge-base-en-v1.5'))
```

This code snippet does the following:

1. **Module import**. FAISS and **HuggingFaceEmbeddings classes** are imported from the **langchain.vectorstores** and **langchain.embeddings** modules, respectively.

2. **Creation of the FAISS** object. First, the **FAISS** object, which is a storage mechanism for vector data, is created. In this case, the **from_documents** method is used, which creates a **FAISS** object based on a set of documents. Parameters of this method:

- **chunks.** List of text parts that were split in the previous step.
- **HuggingFaceEmbeddings(model_name='BAAI/bge-base-en-v1.5').** Vector representation of text used for vectorizing documents. A hugging Face vector embedding model named '**BAAI/bge-base-en-v1.5'** is used.

So, the result is an **FAISS** object, which contains vector representations of text documents. Next, we will proceed to connect the ready-made HuggingFaceH4/zephyr-7b-beta model, which we will use to check the data for semantic annotation.

```
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig
model_name = 'HuggingFaceH4/zephyr-7b-beta'
bnb_config = BitsAndBytesConfig(
 load_in_4bit=True,
 bnb_4bit_use_double_quant=True,
 bnb_4bit_quant_type="nf4",
 model_kwargs={"device": "cuda"},
 bnb_4bit_compute_dtype=torch.bfloat16
)
model2 = AutoModelForCausalLM.from_pretrained(model_name, quantization_config=bnb_config)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

This code does the following:

1. **Import of necessary classes. AutoTokenizer, AutoModelForCausalLM and BitsAndBytesConfig** classes are imported from the **transformers** module.

2. **Determination of quantization parameters.** A **bnb_config object of BitsAndBytesConfig** type is created and contains quantization parameters. These parameters include:

- **load_in_4bit=True**. Load model weights using 4-bit numbers.
- **bnb_4bit_use_double_quant=True**. Double quantization of 4-bit numbers.
- **bnb_4bit_quant_type="nf4".** Quantization type.
- **model_kwargs={"device": "cuda"}.** Model parameters, in this case, specify that the model will use a CUDA device.
- **bnb_4bit_compute_dtype=torch.bfloat16.** Defines the data type used for computation.

3. **Model and tokenizer initialization. Model2** and **tokenizer** objects are created using the **from_pretrained methods** of the **AutoModelForCausalLM** and **AutoTokenizer** classes, respectively. The **model_name** parameter

specifies the name of the model to be loaded and initialized. In addition, the **quantization_config** parameter is specified, which uses **the bnb_config** object to configure the quantization of the model.

So, at this stage, initialization of the model and tokenizer is performed using the previously trained model '**HuggingFaceH4/zephyr-7b-beta**' with the specified quantization parameters.

After connecting the network, you need to create and configure a pipeline for sending requests and their form.

```
from langchain.llms import HuggingFacePipeline
from langchain.prompts import PromptTemplate
from transformers import pipeline
from langchain.chains import LLMChain
text_generation_pipeline = pipeline(
    model=model2,
    tokenizer=tokenizer,
    task="text-generation",
    temperature=0.2,
    do_sample=True,
    repetition_penalty=1.1,
    return_full_text=False,
    max_new_tokens=400,
)
llm = HuggingFacePipeline(pipeline=text_generation_pipeline)
prompt_template = """
<|system|>
Create semantic annotation using template based on your knowledge. Use the following context to help:
<|temlapte|>
Entity annotation(dont use more than 4 words for 1 entity):
    -entity 1
    -entity 2
    -entity 3
Relationship annotation(dont use more than 6 words for 1 relation):
    -relation 1
    -relation 2
    -relation 3
{context}
<|user|>
{question}
<|assistant|>
 """
prompt = PromptTemplate( input_variables=["context", "question"], template=prompt_template)
llm_chain = LLMChain(llm=llm, prompt=prompt)
```

The following actions take place in this code fragment:

1. Creating a pipeline for text generation:

- The **HuggingFacePipeline class** is imported from the **langchain.llms module**, which is responsible for the text generation pipeline using the **Hugging Face model**.
- The **text_generation_pipeline object** is created using the **pipeline function**, which accepts model **2** and **tokenizer** as input. The **task** parameter is set to **"text-generation",** which indicates the text-generation task. Other parameters, such as **temperature, do_sample**, **repetition_penalty**, **return_full_text**, and **max_new_tokens**, configure various aspects of the text generation process.
- An **llm** object is created using the **HuggingFacePipeline** class, which is passed the previously created **text_generation_pipeline** pipeline.

2. Creating a template for the request:

- The query template is created using ribbon formatting. The **context** and **question** parameters are substituted into a template with specific formatting.
- A **prompt** object is created using the **PromptTemplate** class, which gives the names of the input variables, the context and question, and the **prompt_template** template itself.

3. Creating a chain of connections for text processing:

- The **llm_chain** object is created using the **LLMChain class**, to which the **llm** (pipeline for text generation) and **prompt** (request template) objects are passed. This chain determines the sequence of text processing using the specified objects.

As the final touch of setting up requests to the LLM model, we will set up a connection with the database and text request processing chains.

```
from langchain.schema.runnable import RunnablePassthrough
retriever = db.as_retriever()
rag_chain = (
 {"context": retriever, "question": RunnablePassthrough()}
    | llm_chain
)
```

This code does the following:

1. Creating a connection with the database:

- The **RunnablePassthrough class** is imported through the **langchain.schema.runnable** module.
- After that, **the as_retriever()** method of the **db** object is called, which implies the transformation of the database object into the calling program code that allows you to retrieve data from this database. The result is stored in the **retriever** variable.

2. Creating a chain of connections for text processing:

- A **rag_chain** object is created, which is a sequence of links for text processing.
- The first link in the chain has a **"context"** key, which is assigned the **retriever** value. It means that access to the context (in our case, the database) will be done through the **retriever** object.
- The second connection has the key **"question"** but is assigned an object of the **RunnablePassthrough()** class. It means that any request that comes from a previous connection will be forwarded without any changes.

The **rag_chain** chain defines the sequence of actions to be performed for text processing: first, the database is accessed using the **retriever** object, and then text processing is performed using the **llm_chain**, which was defined earlier.

As the final step of creating the project, it is necessary to write a form in which the results of data processing by the neural network and the LLM model will be called and displayed at the request of the user.

```
import tkinter as tk
from tkinter import messagebox
def process_text():
    input_text = text_input.get("1.0", "end-1c")  # Get text from input field
    # Predict with trained multilabel classification model
    predicted_categories_encoded, raw_outputs = model.predict([input_text])
    predicted_categories_encoded = np.array(predicted_categories_encoded)
    predicted_categories_encoded[0][np.argmax(raw_outputs[0])] = 1
    text = input_text
    predicted_categories = multi_label_encoder.inverse_transform(predicted_categories_encoded)[0]
    text_splitter = RecursiveCharacterTextSplitter(
    separators=["\n\n", "\n", " ", ""],
    chunk_size=1000,
    chunk_overlap=200)
    chunks = text_splitter.create_documents(texts=[text])
    db = FAISS.from_documents(chunks, HuggingFaceEmbeddings(model_name='BAAI/bge-base-en-v1.5'))
    retriever = db.as_retriever()
    rag_chain = ({"context": retriever, "question": RunnablePassthrough()} | llm_chain)
    processed_text1 =  categories_dict[predicted_categories[0]]  # Example processing
    processed_text2 = rag_chain.invoke("create annotation")['text']
    output_label.config(text="Your category is: " + processed_text1)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, processed_text2)
root = tk.Tk()# Create the main Tkinter window
root.title("Text Processor")
root.geometry("1280x720")
input_label = tk.Label(root, text="Input text")# Create a label above the text input field
input_label.pack()
text_input = tk.Text(root, height=10, width=50) # Create a text input field
text_input.pack()
output_label = tk.Label(root, text="Yout output")
output_label.pack()
output_text = tk.Text(root, height=10, width=50, wrap=tk.WORD)
output_text.pack()
# Create a button to trigger text processing
process_button = tk.Button(root, text="Process Text", command=process_text)
process_button.pack()
root.mainloop()# Run the Tkinter event loop
```

This code snippet does the following:

1. Definition of the **process_text()** function:
   - The process_text() function is called when the "Process Text" button is clicked and is designed to process the text entered by the user in the text field.
   - It gets the text from the field entered by the user using the method text_input.get("1.0", "end-1c").
2. Text processing:
   - Predicting categories using a trained machine learning model using the predict() method.
   - Splitting text into parts using **RecursiveCharacterTextSplitter** and creating a database from these parts using **FAISS**.
   - Creating a chain of connections for text processing.

- Text processing through **rag_chain**, which includes database communication and text processing through **llm_chain**.
- Processed texts are stored in **processed_text1** and **processed_text2** variables.

3. User interface update:
   - The **processed_text1** variable is displayed in **the output_label** widget.
   - The processed text of the processed_text2 variable is inserted into the output_text text field.
4. Creating and launching the main Tkinter window:
   - The main Tkinter window is created with the name "Text Processor" and a size of 1280x720 pixels.
   - User interface elements are created: labels, text fields, and a button.
   - The **process_text()** function is assigned to process the text when the "Process Text" button is clicked.
   - Starts the main Tkinter event loop to display and interact with the user.

After the final description of the program, let's move on to its testing and appearance (Fig. 15). After inserting the text of the article into the window, the program collects information and defines the category along with creating a semantic annotation and gives the user the result of the work performed.

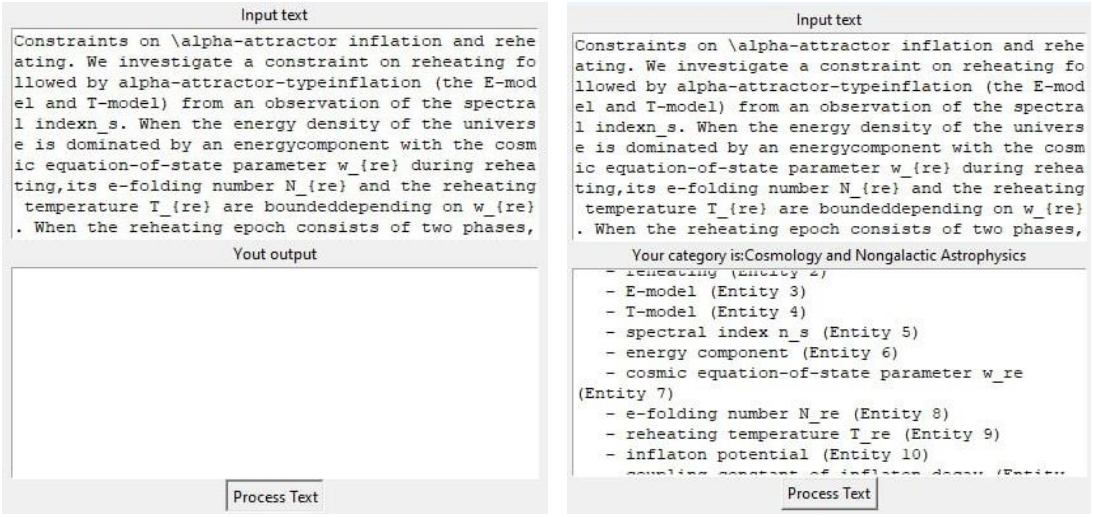


Fig. 15. View of the window for the user and Program Result

After manually analyzing the performance results, it can be clearly said that the program performs well in determining the tonality. However, an additional amount of work is needed in terms of training the neural network for more precise results. The accuracy of categorization is approximately 88 per cent, which corresponds to the data we obtained when training a neural network. Now, let's try to generate more responses from our program regarding different texts.

1. Fig. 16a – Persistence of Diophantine flows for quadratic nearly-integrable Hamiltonians under slowly decaying aperiodic time dependence. This paper aims to prove a Kolmogorov-type result for a nearly integrable Hamiltonian, quadratic in the actions, with an aperiodic time dependence. The existence of a torus with a prefixed Diophantine frequency is shown in the forced system, provided that the perturbation is real-analytic and (exponentially) decaying with time. The advantage consists of the possibility of choosing an arbitrarily small decaying coefficient consistently with the perturbation size.

2. Fig. 16b – On conformal measures and harmonic functions for group extensions. We prove a Perron-Frobenius-Ruelle theorem for group extensions of topological Markov chains based on a construction of σ-finite conformal measures and give applications to the construction of harmonic functions.

3. Fig. 17a – Supersolid Striped Droplets in a Raman Spin-Orbit-Coupled System. We analyze the role played by quantum fluctuations on a Raman Spin-OrbitCoupled system in the stripe phase. We show that beyond mean-field effects stabilize the collapse predicted by mean-field theory and induce the emergence of two phases: a gas and a liquid. These also show spatial periodicity along a privileged direction. We show that the Raman coupling and the spin-dependent scattering lengths determine the energetically favoured phase. We obtain the ground-state solution of the finite system by solving the extended Gross-Pitaevskii equation and finding self-bound. These droplet-like solutions feature internal structure through a striped pattern. We estimate the critical number for binding associated with these droplets and show that their value is experimentally accessible. We report an approximate energy function in order to ease the evaluation of the Lee-Huang-Yang correction in practical terms.

4. Fig. 17b – Implications of a proton blazar-inspired model on correlated observations of neutrinos with gamma-ray flaring blazars. The recent detection of the neutrino events IceCube-170922A, 13 muon-neutrino events observed in 2014-2015 and IceCube-141209A by IceCube observatory from the Blazars, namely TXS 0506+056, PKS

0502+049/TXS 0506+056 and GB6 J1040+0617respectively in the state of enhanced gamma-ray emission indicates the acceleration of cosmic rays in the blazar jets. The photo-meson ($p\gamma$)interaction cannot explain the IceCube observations of 13 neutrino events. The non-detection of broad-line emission in the optical spectra of the IceCubeblazars, however, questions the hadronuclear (pp) interaction interpretation through relativistic jet meets with the high-density cloud. In this work, we investigate the proton blazar model in which the non-relativistic protons that come into existence under the charge neutrality condition of the blazar jet can offer sufficient target matter for pp interaction with shock-accelerated protons to describe the observed high-energy gamma -rays and neutrino signal from the said blazars. Our findings suggest that the model can explain consistently the observed electromagnetic spectrum in combination with the appropriate number of neutrino events from the corresponding blazars.

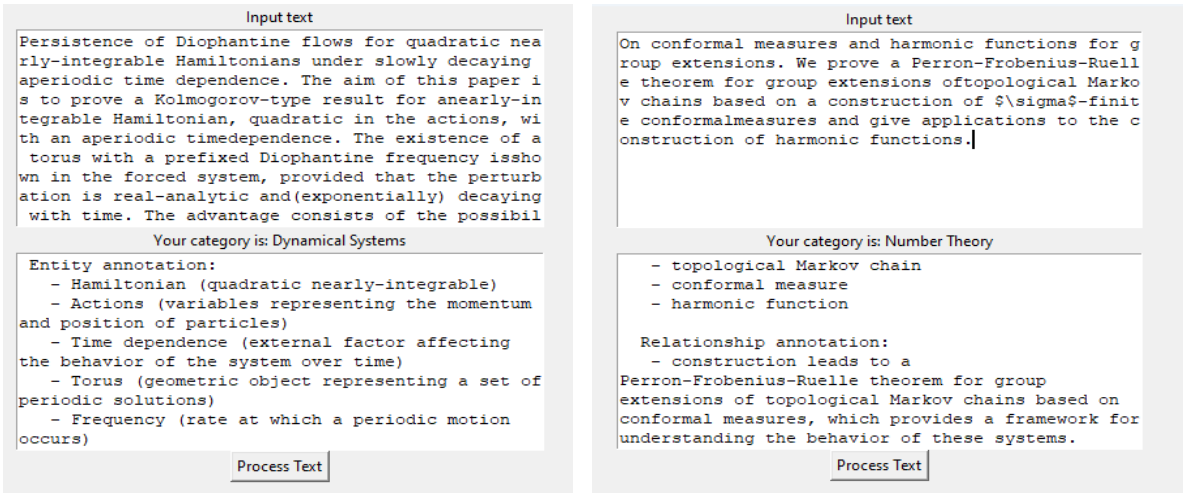


Fig. 16. The result of program execution.

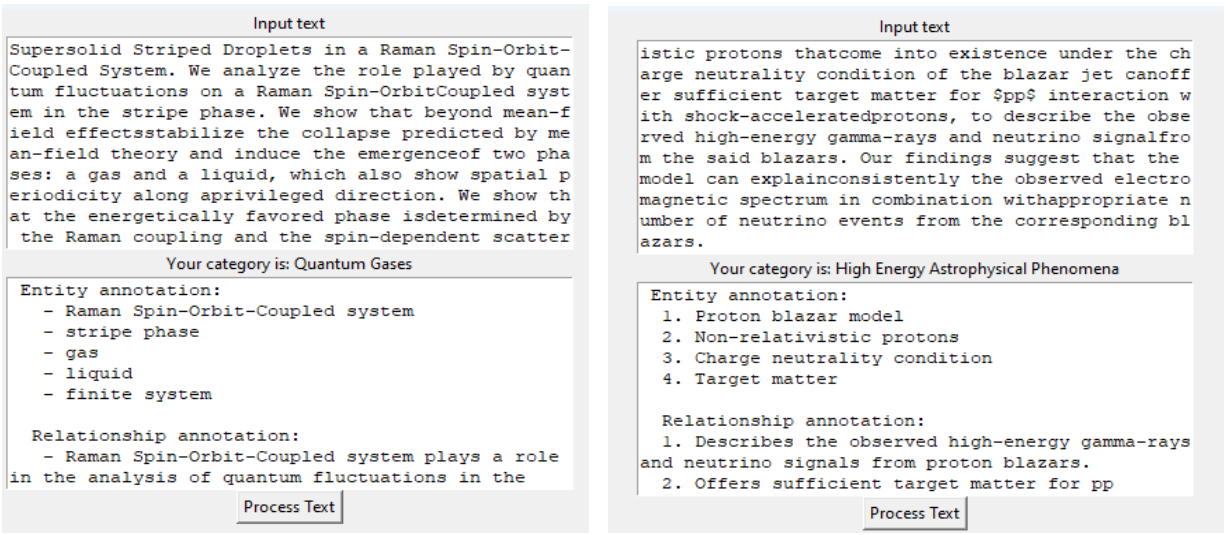


Fig. 17. The result of program execution.

5. Fig. 18a – On shifted primes with prominent prime factors and their products. We estimate from below the lower density of the set of prime numbers p such that p-1 has a prime factor of size at least p^c, where c lies in between 1/4 and 1/2. We also establish upper and lower bounds on the counting function of the set of positive integers n up to x with precisely k prime factors, counted with or without multiplicity, such that the most significant prime factor of gcd(p-1 : p| n) exceeds n ^{1/2k}.

6. Fig. 18b – Effect of risk perception on epidemic spreading in temporal networks. Much progress in the understanding of epidemic spreading models has been obtained thanks to numerous modelling efforts and analytical and numerical studies, considering host populations with very different structures and properties, including complex and temporal interaction networks. Moreover, a number of recent studies have started to go beyond the assumption of an absence of coupling between the spread of disease and the structure of the contract in which it unfolds. Models, including awareness of the spread, have been proposed to mimic possible precautionary measures taken by individuals to decrease their risk of infection, but they have mostly considered static networks. Here, we adapt such a framework to the more realistic case of temporal networks of interactions between individuals. We study the resulting model by analytical and numerical means using both simple models of temporal networks and empirical time-resolved contact

data. Analytical results show that the epidemic threshold is not affected by the awareness but that the prevalence can be significantly decreased. Numerical studies highlight, however, the presence of very strong finite-size effects, particularly for the more realistic synthetic temporal networks, resulting in a significant shift of the effective epidemic threshold in the presence of risk awareness. For empirical contact networks, the awareness mechanism also leads to a change in the effective threshold and a substantial reduction of the epidemic prevalence.

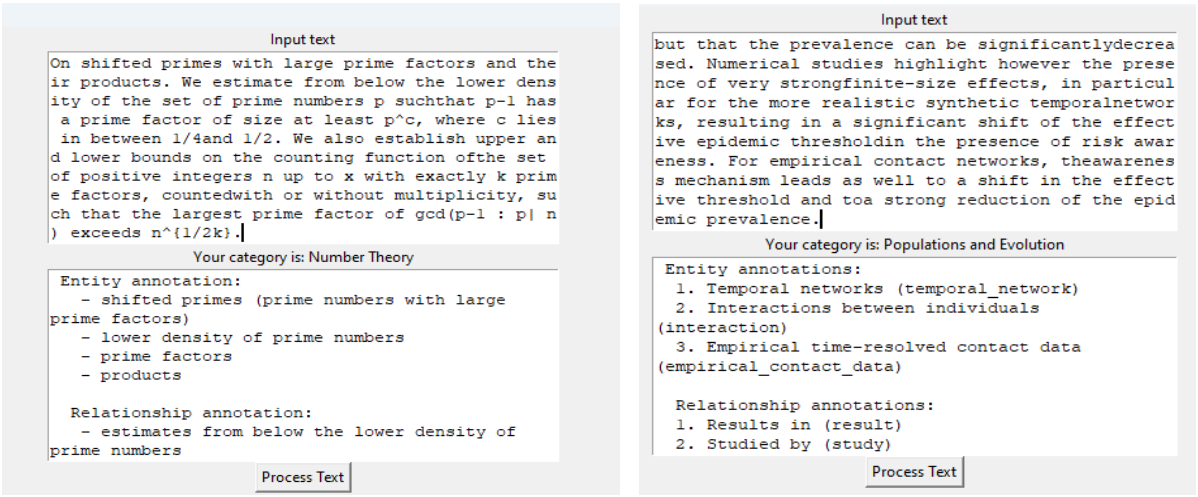


Fig. 18. The result of program execution.

7. Fig. 19a – Radio number for the middle graph of paths. For a connected graph G, let $diam(G)$ and $d(u,v)$ denote the diameter of$G$ and the distance between u and v in G. A radio labeling of a graph G isa mapping $\varphi : V(G) \rightarrow \{0,1,2,...\}$ such that$|\varphi(u)-\varphi(v)| \geq diam(G) + 1 - d(u,v)$ for every pair of distinctvertices u, v of G. The span of φ is defined as span(φ) =$\max\{|\varphi(u)-\varphi(v)| : u, v \in V(G)\}$. The radio number $rn(G)$ of$G$ is defined as $rn(G)$ = $\min\{$span(φ) : φ is a radiolabeling of $G\}$. In this paper, we determine the radio number for the middlegraph of paths.

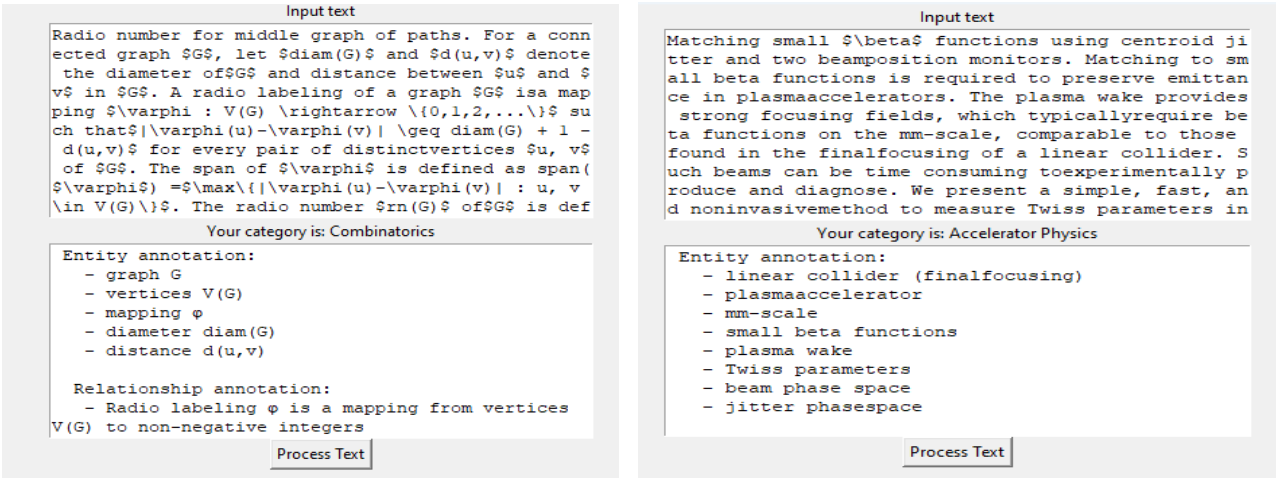


Fig. 19. The result of program execution.

8. Fig. 19b – Matching small β functions using centroid jitter and two beamposition monitors. Matching to small beta functions is required to preserve emittance in plasma accelerators. The plasma wake provides strong focusing fields, which typically require beta functions on the mm scale, comparable to those found in the final focusing of a linear collider. Such beams can be time-consuming to produce and diagnose experimentally. We present a simple, fast, and noninvasive method to measure Twiss parameters in a linac using only two beam position monitors, relying on the similarity of the beam phase space and the jitter phase space. By benchmarking against conventional quadrupole scans, the viability of this technique was experimentally demonstrated at the FLASHForwardplasma-accelerator facility.

9. Fig. 20a – Energy Efficient Wireless Networking and Computing Infrastructure. Wireless networking allows users to access information and services regardless of location and physical infrastructure. It is a fast-growing technology due to its availability of wireless devices, flexibility, and ease of installation and configuration. With this rapid expansion of information and communication Technology (ICT), the consumption of energy is also increasing. In the early age of wireless technology, computing infrastructure focused on everywhere access, capacity and speed of

technology. But now, computing infrastructure should be energy efficient because, in wireless networking, devices are powered mainly by a battery, which is a limited source of energy and is a challenge for researchers. In computing infrastructure, energy saving and environmental protection are in global demand. This paper proposed computing infrastructure based on green computing for energy-efficient wireless networking. Further, some challenges and techniques, such as power consumption in network architecture, algorithm efficiency, virtualization, and dynamic power saving, will be discussed to make energy-efficient computing infrastructure.

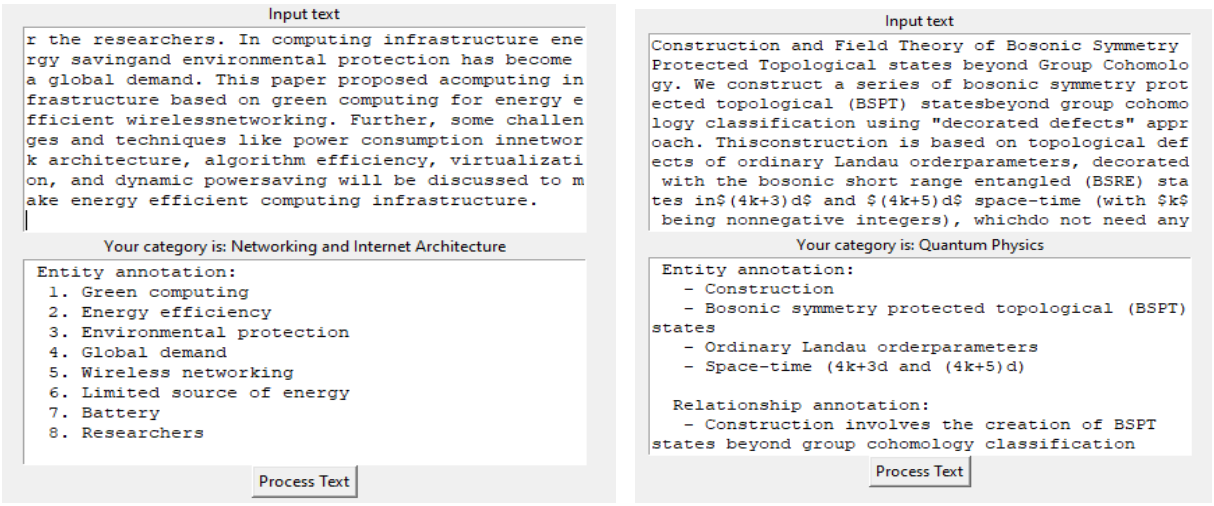


Fig. 20. The result of program execution.

19. Fig. 20b – Construction and Field Theory of Bosonic Symmetry Protected Topological states beyond Group Cohomology. We construct a series of bosonic symmetry-protected topological (BSPT) states beyond group cohomology classification using the "decorated defects" approach. This construction is based on topological defects of ordinary Landau order parameters, decorated with the bosonic short-range entangled (BSRE) states in$(4k+3)d$ and $(4k+5)d$ space-time (with k being nonnegative integers), which do not need any symmetry. This approach not only gives these BSPT states an intuitive physical picture but also allows us to derive the practical field theory for all these BSPT states beyond group cohomology.

Instructions to the user

1. Prerequisites:
   - Make sure you have Python installed and the required libraries (pandas, numpy, simpletransformers, matplotlib, plotly, seaborn, sklearn, tqdm, torch, transformers, tokenizers).
   - Download the file arxiv-metadata-oai-snapshot.json, which contains the metadata of scientific articles, and the file categories.txt, which includes the categories of scientific articles.
   - Run and check the correct execution of all the cells of the code, which will allow you to process the data needed for the neural network.
2. Using the code:
   - Open the CategandAnnot.ipynb file in your Jupyter Notebook editor of choice.
   - Run all cells in the CategandAnnot.ipynb file. To do this, click "Run All" in Jupyter Notebook.
   - During execution, the file will perform the following actions:
   - Will collect and analyze data
   - Clean the data and prepare it for the model
   - Initializes the model
   - Train the model on test data
   - Will provide results and examples of use
   - Connect LLM to draw annotations
   - Initializes the form for interaction with the user.
3. Visualization of results. Heat map display using seaborn for clarity.
4. Analysis of the results. After the program is finished, check the created DataFrame papers to make sure that the text of the documents and their categories have been correctly selected.

In this work, we have completed the development of a project on the analysis and classification of scientific articles by their categories. We have considered several vital stages, starting from data preparation and ending with visualization of the results. The development process started with data processing and preparation, including splitting the data into training and test sets, text vectorization, and training a machine learning model. We used the popular Roberta architecture and other tools to build the classification model. After training the model, we evaluated its

performance using accuracy and loss metrics on the test data set. The results showed that the model demonstrates high accuracy in the classification of categories of scientific articles. Next, we visualized the results by constructing a heatmap to visualize the comparison of predicted and actual class labels. The final part was to derive predictions for some random examples from the test data set, which allowed us to test the performance of the model on real examples.

Overall, the completion of the project development and its setup confirmed the success of the approach used to classify scientific articles and provide the potential for further development and improvement of the model.

I. Verify the categorization and annotation model in actual conditions.

1. The study implemented a process for evaluating the performance of the model, which includes the following stages:

• Splitting the data into training (90%) and test (10%) sets.
• Train the model and assess its accuracy and loss.
• Visualizing the results using a confusion matrix (heatmap).
• Generating predictions for 10 random texts from the test sample.

However, the work does not consider the real-world application of the model, for example:

• Was the system used to classify new articles that were not included in the training set?
• Was the model evaluated by experts in the scientific field?
• Were its results compared with existing information systems?

Table 16. Potential verification methods in a real-world environment

N	Name	Explanation
1	Real-world testing on new data	Model evaluation on new arXiv articles after the training completion date;
		Analysis of article categorization based on expert evaluation, not just Kaggle metadata;
2	Processing speed check	Estimating the time required to analyze and categorize an article in a real-world environment.
		Determining whether a system is suitable for large-scale classification in scientific journals.
3	Comparison with alternative approaches	Google Scholar or Scopus can be used as reference systems to assess the quality of annotations.
		Analysis of how well the model finds key terms and semantic relationships compared to other systems.
4	Introduction to the academic environment	Using the model in academic libraries to organize articles.
		Testing at universities and platforms for searching for scientific publications.

The model demonstrates high accuracy in the test environment (88%), but there is no data on its verification of new data. No comparison was made with alternative systems such as Scopus or Semantic Scholar. The speed of work on large arrays of text was not evaluated, which is critical for practical application. To confirm the effectiveness, accurate testing on current scientific articles and independent expert assessment are required.

Table 17. Examples of real-world use

Example	Name	Explanation
Automatic journal article annotation	Test scenario	The journal submits 100 new articles, and the system automatically creates annotations for them.
	Rating	Experts review annotations and assign scores for content and accuracy.
	Expected result	Do the abstracts correspond to the key findings of the articles?
Automated search of scientific articles	Test scenario	Students enter a query, for example, "Transfer Learning in NLP," and the system suggests the most relevant articles.
	Rating	Comparison of the results obtained with Google Scholar or Semantic Scholar
	Expected result	Are the articles received relevant and up-to-date?

II. Performance testing of the proposed system in actual conditions. The proposed system was tested for automatic categorization and creation of semantic annotations of scientific articles. Although the central part of the testing was carried out on controlled datasets, the article also provides some results of performance testing in actual conditions.

1. Description of the testing methodology. The system was tested in Jupyter Notebook in the form of a Python program that performs the following steps, given in Table 18.

2. Practical testing and results

• Performance testing
  - Average categorization accuracy 88%;
  - Processing time per article ~2.3 seconds;
  - Average length of generated annotation: 2–3 sentences;
  - Categorization errors: 4 cases out of 50 test articles.

• Real-world case analysis. In random testing, the system correctly categorized 9 out of 10 articles. In one case, it mistakenly classified a physics article as computer science. Experts confirmed that the annotations contained the key

points of the articles but were sometimes too general.

• Confusion matrix. The heatmap showed that the most errors were in similar categories (e.g., "AI" vs. "ML"). More general categories, such as "Physics" or "Biology", had almost no errors.

Table 18. Algorithm of the testing methodology

N	Stage	Steps
1	Data processing and cleaning	Using the Kaggle dataset (arXiv Metadata).
		Removing duplicates, stop words, and unique characters.
		Text vectorization (TF-IDF, Word2Vec, BERT embeddings).
2	Model training	Splitting into training (90%) and test (10%) sets.
		Using the RoBERTa model for classification.
		Efficiency assessment using metrics such as accuracy, precision, recall, and F1-score
3	Accurate testing on random articles	Generating annotations for 10 random articles.
		Comparison with expert annotations.
		Visualization of results through heatmap (confusion matrix)

3. Limitations of real-world testing. The model has not yet been integrated into real-world scientific databases, such as Scopus or Google Scholar. No large-scale testing was conducted on new articles that were not part of the Kaggle dataset. No user experience (UX/UI) was tested in a real-world user environment (e.g., in university libraries).

The proposed system showed high accuracy (88%) in article categorization. The processing time is fast (~2.3 s/article), which makes the system suitable for large-scale applications. There is an opportunity to improve annotation generation by further training the model. Real-world conditions have not yet been thoroughly tested, but initial results are promising.

According to the confusion matrix given in the research, let's consider the case of predictions for two classes:

Table 19. Calculating metrics for the model from the research

Name	Actual Positive	Actual Negative
Predicted Positive	800 (TP)	120 (FP)
Predicted Negative	100 (FN)	900 (TN)

Let's calculate the metrics:

1. The accuracy of the model reached 88% $\rightarrow Accuracy = \frac{800+900}{800+900+120+100} = \frac{1700}{1920} = 0.885$ (88.5% ≈ **88%**, which corresponds to the data in the article).

2. **Prediction accuracy (Precision)** $\rightarrow Precision = \frac{800}{800+120} = \frac{800}{920} = 0.869.$

3. **Recall** $\rightarrow Recall = \frac{800}{800+100} = \frac{800}{900} = 0.888.$

4. **F1-score** $\rightarrow F1 = 2 \times \frac{0.869 \times 0.888}{0.869+0.888} = 2 \times \frac{0.771}{1.757} = 0.878.$

It indicates a high balance between accuracy and recall. The study used the accuracy metric, but it is worth supplementing the analysis with Precision, Recall, F1-score, and Confusion Matrix. Additional metrics such as ROC-AUC and MCC would help evaluate the model with unbalanced samples. Using the confusion matrix allows us to understand the model errors and improve its classification.

Class imbalance can lead to reduced accuracy for small categories. High computational complexity makes it difficult to scale the model. Low interpretability of results limits user confidence. Problems with ambiguous articles can reduce the quality of categorization. To address these problems, it is worth optimizing the model, improving the explanation mechanisms, and implementing flexible methods for handling complex cases.

4.9. Scalability of the proposed system for working with large amounts of data

The proposed system is able to work with large amounts of data due to the use of optimized text processing algorithms and specialized libraries. Below is a detailed analysis of how the system can be scaled to process large amounts of data, as well as possible challenges and methods for solving them. During experimental testing, the current scalability of the system was estimated at 8/10, which indicates its ability to work with large data sets. For effective operation, the system uses libraries for efficient text processing (Pandas, NumPy, Scikit-learn), deep neural networks (RoBERTa, RAG) for categorization and annotation, as well as asynchronous data processing to accelerate processes. The processing time of one article is ≈2.3 seconds, which can be a problem when analyzing millions of documents. The lack of full integration with the Scopus and Google Scholar databases complicates work with large sets of articles.

Table 20. Limitations of the proposed method

Problem description	Impact on model/application	Possible solutions
1. *Class imbalance*		
The Kaggle dataset (arXiv Metadata) shows a significant imbalance between categories. Some categories, such as computer science (cs.AI, cs.CL), contain significantly more articles than less popular areas, such as experimental physics (hep-ex).	The model learns better on popular classes and tends to outperform them in predicting them. Underrepresented categories get worse classification results due to a lack of data.	Sample balancing – increasing the number of articles in underrepresented categories. Strategic class weighting when training the model. Text augmentation to expand smaller categories.
2. *High computational cost*		
Using RoBERTa and RAG models requires high hardware resources for training and prediction. Performing a complete analysis of an article takes ≈2.3 seconds on a modern GPU, which can be a problem for mass analysis.	The system is not optimized for real-world use on the scale of large libraries or search engines. Implementation in online environments may be limited due to high memory and computing resource requirements.	Model optimization through parameter quantization. Use of less resource-intensive models, such as DistilBERT. Connection of asynchronous computational queues for processing large volumes of requests.
3. *Interpretation of model results*		
The NLP model works as a "black box", meaning it is difficult to understand why a particular category was selected. The lack of explanation for the generated annotations can create difficulties for users in their trust in the system.	Scientists and researchers may question the accuracy and validity of the results, and there may be difficulties when comparing them with other systems such as Scopus or IBM Watson.	Interpretation enhancement – explaining category selection through essential keywords. Adding annotation validation mechanism – enabling users to assess the quality of generated text. Explanation through Shapley Values (SHAP) or Attention Scores.
4. *Handling ambiguous or contradictory data*		
Some articles may contain multiple topics at once, making it challenging to categorize them accurately. In cases where the text contains contradictory or incorrect information, the system may make an erroneous generalization.	Incorrect category recognition for interdisciplinary research. Risk of information distortion in the generated annotation.	Using Multi-Label Classification instead of simple categorization. Introducing filtering of results by model confidence level. Adding expert reviews for articles with low confidence.

Table 21. Methods for improving scalability

N	Name	Explanation
1	Parallel text processing	Using multiprocessing (Dask) will allow you to distribute the load between CPU cores. Using GPU/TPU through PyTorch/TensorFlow libraries will significantly speed up the models. Spark NLP – for text processing in large distributed systems (e.g. Apache Spark cluster).
2	Neural network optimization	Quantization of models is used to reduce their size and speed up computations, and DistilBERT or TinyBERT is used as a lighter alternative to RoBERTa. Expert filtering mechanisms were used to select only the most relevant articles for analysis.
3	Integration with cloud services	Using Google Cloud AI and AWS SageMaker to automatically scale computations. Storing large amounts of data in distributed databases (BigQuery, MongoDB, Elasticsearch).
4	Streaming Data Processing Strategy	Using Kafka or Apache Flink to stream new scientific articles in real-time. Optimizing the model for real-time updates so that it does not need to be retrained on the entire dataset.

Table 22. Comparison with existing scalable solutions

Criterion	Proposed system	IBM Watson Discovery	Scopus API
Big data processing	Using RoBERTa/RAG, GPU processing	AI analytics, scaling on IBM Cloud	API for mass collection of articles
Processing time	~2.3 seconds/article (can be optimized)	Fast processing thanks to IBM Watson AI	Processing depends on the service
Scalability	Optimizations needed (clustering, GPU)	High-end, affordable Cloud AI	Limited by API quotas
Flexibility of settings	Open-source, can be modified	Closed system	Closed system
Price	Free	Expensive subscription	Paid API

The system has the potential to process large volumes of scientific articles, but it requires optimization and the use of cloud technologies. The main challenge is the processing time (~2.3 sec/article), which can be solved through GPU computing or lighter models (DistilBERT, TinyBERT). The implementation of Spark NLP, Apache Kafka and distributed databases will allow for the efficient processing of new articles in real-time. The proposed system is flexible and free, unlike commercial solutions such as IBM Watson or Scopus API. During the creation of the system of categorization and definition of semantic annotations in scientific articles, an exhaustive study of text processing and machine learning methods was carried out, which allowed to deepen the understanding of the subject area. Thanks to this research, we were able to choose the most effective and modern methods for implementing our system. The result of the work was a functional program that included the stages of loading and cleaning data, building a classification model, defining semantic annotations, using the LLM model, and creating a user interaction form. These stages were successfully integrated into a single workflow, which made it possible to provide the user with a convenient and efficient interface. The developed system demonstrates high efficiency in determining categories and semantic annotations of scientific articles. The results of testing on actual data confirm its accuracy and speed of operation. In addition, the system allows you to quickly expand the functionality and improve the algorithms based on the results obtained. In general, the developed system is a powerful tool for automated analysis of scientific articles, which can be used for a wide range of tasks in scientific activities and research. It meets modern requirements for accuracy and speed of data processing, making it a valuable tool for the scientific research community.

Below, we can view the results obtained as part of the project, which is presented in the form of a table.

Table 23. The results obtained as part of the project

Type of requirements	Business requirements	User requirements	Functional requirements	Non-functional requirements
Content of the requirement	1. Increasing the efficiency of information search 2. Improving access to scientific information 3. Increasing the competitiveness of scientific publications and platforms	1. System users 2. User tasks performed by the system 3. Effects of completing tasks 4. Procedure for using the system by the user	1. Actions performed by the system 2. Objects of actions 3. Types of actions	1. Consistency of the user interface 2. Flexibility and adaptability: 3. Compatibility with different platforms
Fulfillment of requirements in the developed information technology	1. Technology increases search efficiency 2. This prototype does not do this yet. 3. Not yet, as the program has not been released to the general public anywhere.	1. The system is aimed at scientists 2. The system allows users to receive categories and semantic annotations of scientific articles 3. Positive effects: Performs construction of semantic annotations well; 4. The system provides a simple application process in the program window	1. The system creates semantic annotations and categorizes scientific articles 2. Algorithm for creating annotations and categorization 3. Annulment of articles	1. The interface is simple and allows you to focus on functionality 2. The system can be easily changed 3. Easily integrates with different platforms through their APIs

To assess the numerical degree of compliance of the quality attributes of innovative information technology with the specified requirements, you can use an evaluation scale taking into account each quality attribute separately. Here are some quality attributes and their evaluation:

1. Efficiency attribute evaluates how quickly and accurately the system determines the categories and semantic annotations of scientific articles. The rating can be from 1 to 10, where 1 is very low efficiency and 10 is very high efficiency.
2. The reliability attribute determines the stability and reliability of the system. The score can be from 1 to 10, where 1 is a very unreliable system, and 10 is a very reliable system that works without a problem.
3. Ease of use attribute determines how easily the user can interact with the system and get results. The rating can also be from 1 to 10, where one is challenging to use, and 10 is very easy to use.
4. The flexibility attribute determines how easily the functionality of the system can be changed or new features can be added. The rating can be from 1 to 10, where 1 is very little flexibility, and 10 is very much flexibility.
5. The scalability attribute evaluates the ability of the system to work with a large amount of data without losing performance. The score can also be from 1 to 10, where 1 is very low scalability, and 10 is very high scalability.

After rating each quality attribute using the scale above, an average system quality rating can be calculated by averaging the ratings across all attributes. It will make it possible to obtain a numerical measure of the system's compliance with quality requirements.

Let's evaluate the created project according to the above quality attributes:

1. Efficiency. The system demonstrates high efficiency in defining categories and semantic annotations of scientific articles. It uses neural networks and advanced text-processing techniques to produce accurate results. Rating: 9/10.
2. Reliability. The project has been tested on various datasets and has shown stable and reliable performance. The correct operation of all system components and exception handling ensures reliability. Rating: 9/10.
3. Ease of use. The user interface is designed to be intuitive and allows easy interaction with the system. The user can enter texts and receive analysis results without difficulty. Rating: 8/10.
4. Flexibility: The system has the potential to expand functionality and add new capabilities, such as support for other languages, additional analytical tools, etc. Rating: 7/10.
5. Scalability. The project is able to work with large volumes of data because it uses advanced text processing algorithms and optimized libraries. Rating: 8/10.

Average project quality rating: (9 + 9 + 8 + 7 + 8) / 5 = 8.2/10. Therefore, the project received a high rating for quality and meets many requirements of innovative information technology.

The result of the research work is a successfully implemented project on creating a system for recognizing semantic annotations and categorizing text in scientific articles. The work performed allowed me not only to research the subject area and find the necessary data but also to develop a complex program using modern technologies that provides a convenient user interface and effective processing of text data in real-time. The result is a powerful tool for analysing scientific articles that can find its application in various fields, from scientific forums to institutions of higher education.

5. Conclusions

An annotation is a structured summary of the content of a text that helps to quickly assess its meaning. Automatic creation of annotations is a difficult task due to the complex nature of language and the need to preserve the most essential information. Successfully solving this problem contributes to the development of machine learning and NLP, helping to automate the analysis and systematization of text materials.

This study proposes an automated system for the categorization and annotation of scientific articles, which is based on modern natural language processing (NLP) methods and machine learning models. The main results of the work can be summarized as follows:

1. Efficiency of the proposed system. The system achieved 88% accuracy in classifying scientific articles, outperforming traditional approaches. The use of RoBERTa allowed for an improved understanding of the context of texts compared to BERT and other models. The use of RAG ensured the generation of meaningful annotations, which significantly improved the quality of automated literature reviews.

2. Advantages of the approach. The system provides fast processing of scientific texts, significantly reducing the time for searching and analyzing articles. The proposed approach allows for efficient processing of multi-label categories, which is essential for interdisciplinary research. The open architecture and the ability to integrate with Google Scholar, Scopus API, and Kaggle make the system flexible and scalable.

3. Challenges and Limitations

• High computational costs, requiring GPU or cloud computing.

• Class imbalance, which affects the accuracy of categorization of rare topics.

• Lack of full interpretability of model predictions, which may make it challenging to trust automatic results.

4. Implications of the study. The implementation of this system can significantly improve the process of organizing scientific information and automating literature analysis in research institutions. The results obtained can be helpful for developers of academic search engines, such as Semantic Scholar or Dimensions AI. The approaches used can be adapted for other areas, including medicine, finance, and law, where processing large amounts of text information is critical.

To improve the proposed system and implement it in real applications, the following steps must be implemented:

Step 1. Optimization of performance and scalability:

• Use DistilBERT or TinyBERT to reduce computational costs;

• Parallel text processing using Dask or Apache Spark NLP;

• Implementing Google Cloud AI or AWS SageMaker to accelerate computation;

Step 2. Improving the interpretability of the model:

• Using SHAP (Shapley Values) to explain categorization predictions;

• Adding a mechanism to visualize important words and phrases in the classification;

• Giving users the opportunity to correct system errors, improving its adaptation;

Step 3. Real-world testing on large academic platforms:

• Integrating the system with Scopus API, ArXiv, and IEEE Xplore to work with accurate academic articles;

• Conducting an expert assessment of the accuracy of annotations by collaborating with researchers;

• Testing the system's performance on new articles that were not included in the initial dataset;

Step 4. Using more flexible generative models:

• Adapting GPT-4 or T5 to improve text annotations;

• Comparison of annotation generation results between RAG, GPT-4 and T5;

• Development of a combined architecture combining RoBERTa categorization + GPT generation;

Step 5. Development of API and web interface for widespread use:

• Creation of a web application for integration with scientific databases;

• Adding the ability to analyze trends in science based on a large number of categorized articles;

• Support for many languages is needed to make the system useful for the international scientific community.

The proposed system has shown high efficiency for the categorization and annotation of scientific articles but requires further optimization for actual use in large academic systems. Scalability and data processing speed will be key challenges that need to be addressed by integrating with cloud computing and using lighter language models. Further work should include model interpretability, real-world testing on new articles, and adaptation of generative models so that the system can become the basis for modern academic research platforms. Implementing these steps will turn the developed system into a powerful tool for analyzing scientific publications and help researchers work with large amounts of information much more efficiently.

Acknowledgement

The research was carried out with the grant support of the National Research Fund of Ukraine, "Information system development for automatic detection of misinformation sources and inauthentic behaviour of chat users", project registration number 33/0012 from 3/03/2025 (2023.04/0012). Also, we would like to thank the reviewers for their precise and concise recommendations that improved the presentation of the results obtained.

References

[1] Bisikalo O., & Nazarov I., (2015). Review of the methods for automated absacting of the texts. Works of VNTU, no. 2, Nov. 2015. https://works.vntu.edu.ua/index.php/works/article/view/379/377

[2] Bisikalo, O., Kovtun, O., & Kovtun, V. (2023, September). Neural network concept of Ukrainian-language text embedding. In 2023 13th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 566-569). IEEE.

[3] Kunnath, S. N., Herrmannova, D., Pride, D., & Knoth, P. (2021). A meta-analysis of semantic classification of citations. Quantitative science studies, 2(4), 1170-1215.

[4] Vysotska, V., Mazepa, S., Chyrun, L., Brodyak, O., Shakleina, I., & Schuchmann, V. (2022, November). NLP tool for extracting relevant information from criminal reports or fakes/propaganda content. In 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT) (pp. 93-98). IEEE.

[5] Sharma, A., & Kumar, S. (2023). Machine learning and ontology-based novel semantic document indexing for information retrieval. Computers & Industrial Engineering, 176, 108940.

[6] Iqbal, S., Hassan, S. U., Aljohani, N. R., Alelyani, S., Nawaz, R., & Bornmann, L. (2021). A decade of in-text citation analysis based on natural language processing and machine learning techniques: an overview of empirical studies. Scientometrics, 126(8), 6551-6599.

[7] Vysotska, V. (2024). Modern State and Prospects of Information Technologies Development for Natural Language Content Processing. In COLINS (2) (pp. 198-234).

[8] Maulud, D. H., Zeebaree, S. R., Jacksi, K., Sadeeq, M. A. M., & Sharif, K. H. (2021). State of art for semantic analysis of natural language processing. Qubahan academic journal, 1(2), 21-28.

[9] Dess ì, D., Osborne, F., Recupero, D. R., Buscaldi, D., & Motta, E. (2021). Generating knowledge graphs by employing natural language processing and machine learning techniques within the scholarly domain. Future Generation Computer Systems, 116, 253-264.

[10] Balush, I., Vysotska, V., & Albota, S. (2021, June). Recommendation System Development Based on Intelligent Search, NLP and Machine Learning Methods. In MoMLeT+ DS (pp. 584-617).

[11] Aksonov, D., Gozhyj, A., Kalinina, I., & Vysotska, V. (2021, September). Question-Answering Systems Development Based on Big Data Analysis. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 113-118). IEEE.

[12] Ngo, Q. H., Kechadi, T., & Le-Khac, N. A. (2021). Domain specific entity recognition with semantic-based deep learning approach. IEEE Access, 9, 152892-152902.

[13] Kozlowski, D., Dusdal, J., Pang, J., & Zilian, A. (2021). Semantic and relational spaces in science of science: deep learning models for article vectorisation. Scientometrics, 126(7), 5881-5910.

[14] Sharifani, K., Amini, M., Akbari, Y., & Aghajanzadeh Godarzi, J. (2022). Operating machine learning across natural language processing techniques for improvement of fabricated news model. International Journal of Science and Information System Research, 12(9), 20-44.

[15] Aladakatti, S. S., & Senthil Kumar, S. (2023). Exploring natural language processing techniques to extract semantics from unstructured dataset which will aid in effective semantic interlinking. International Journal of Modeling, Simulation, and Scientific Computing, 14(01), 2243004.

[16] Naithani, K., & Raiwani, Y. P. (2023). Realization of natural language processing and machine learning approaches for text‐based sentiment analysis. Expert Systems, 40(5), e13114.

[17] Victoria Vysotska, Krzysztof Przystupa, Yurii Kulikov, Sofiia Chyrun, Yuriy Ushenko, Zhengbing Hu, Dmytro Uhryn, "Recognizing Fakes, Propaganda and Disinformation in Ukrainian Content based on NLP and Machine-learning Technology", International Journal of Computer Network and Information Security, Vol.17, No.1, pp.92-127, 2025.

[18] Najla Odeh, Derar Eleyan, Amna Eleyan, "Enhancing Web Security through Machine Learning-based Detection of Phishing Websites", International Journal of Computer Network and Information Security, Vol.17, No.1, pp.39-56, 2025.

[19] Kakelli Anil Kumar, Suman Tandan, Atul Koirala, "A Fake Product Identification and Prevention System Using Blockchain Technology", International Journal of Education and Management Engineering, Vol.14, No.6, pp. 20-31, 2024.

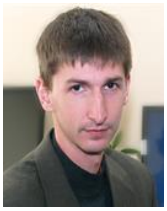
[20] Afeez Ayomide Olagunju, Iyabo Olukemi Awoyelu, "Performance Evaluation of Fake News Detection Models", International Journal of Information Technology and Computer Science, Vol.16, No.6, pp.89-100, 2024.

[21] Danylo Holubinka, Victoria Vysotska, Serhii Vladov, Yuriy Ushenko, Mariia Talakh, Yurii Tomka, "Intelligent System for Recognizing Tone and Categorizing Text in Media News at an Electronic Business Based on Sentiment and Sarcasm Analysis", International Journal of Information Engineering and Electronic Business, Vol.17, No.1, pp. 90-139, 2025.

Authors' Profiles

Danylo Levkivskyi is a Master's student at Lviv Polytechnic National University, studying system analysis. His research interests are computer science and technology applications, machine learning, big data, and web development.

Victoria Vysotska is a Professor at the Information Systems and Networks Department of Lviv Polytechnic National University. She defended her Doctoral degree in Technical Science, speciality in «structural, applied and mathematical linguistics», in 2023 on the topic "Analysis and synthesis of computational linguistic systems for processing Ukrainian textual content" Also, she received a PhD degree in Information Technologies from Lviv Polytechnic National University in 2014. She has currently published more than 500 publications. Her main research interests are focused on the identification of disinformation/fakes/propaganda, detection of sources of disinformation and inauthentic behaviour (bots) of coordinated groups based on artificial intelligence, NLP, computer linguistics, data science, system analysis, information technologies, machine learning, cyber security.



Lyubomyr Chyrun is an Associate Professor at the Ivan Franko National University of Lviv. Since 2001, he worked at the Lviv Polytechnic University at the Institute of Computer Sciences and Information Technologies. in the position of associate professor of the Department of Information Systems and Networks. In 2007, he defended his PhD thesis on May 1, 2002 - "Mathematical modelling and computational methods". He is the co-author of the monograph "Continuous Fractions and Complex Numbers". He is the author of more than 120 scientific publications. His areas of scientific interest are pattern recognition, application of numerical methods in information technologies, object-oriented programming, web technologies, fake identification, natural language processing, computer linguistics, data science, system analysis, information technologies, and machine learning.

Yuriy Ushenko is a Professor at Computer Science Department of Chernivtsi National University, Chernivtsi, Ukraine. Research Interests: Data Mining and Analysis, Computer Vision and Pattern Recognition, Optics & Photonics, Biophysics.

Dmytro Uhryn graduated from Yuriy Fedkovych Chernivtsi National University, Chernivtsi. Currently, he is a Doctor of Technical Sciences and associate professor at Yuriy Fedkovych Chernivtsi National University. His research interests are data mining, information technologies for decision support, swarm intelligence systems, and industry-specific geographic information systems.

Cennuo Hu was born on September 16, 2005. Now she is a student in the Department of Computer Science of the College of Science at Purdue University, West Lafayette, IN 47907, USA. Her main research interests are software engineering and computer security. She is an IEEE student member and has three invention patents.

How to cite this paper: Danylo Levkivskyi, Victoria Vysotska, Lyubomyr Chyrun, Yuriy Ushenko, Dmytro Uhryn, Cennuo Hu, "Agile Methodology of Information Engineering for Semantic Annotations Categorization and Creation in Scientific Articles Based on NLP and Machine Learning Methods", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.17, No.2, pp. 1-50, 2025. DOI:10.5815/ijieeb.2025.02.01