

SCIENTIFIC ARTICLE CLUSTERING AND CLASSIFICATION USING WINNOWING AND GRAPH NEURAL NETWORKS

A PROJECT REPORT

Submitted by

KRISHNA PRASAD H 310621104076

MANISHA R 310621104091

RAHUL R 310621104127

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



EASWARI ENGINEERING COLLEGE, CHENNAI

(Autonomous Institution)

affiliated to

ANNA UNIVERSITY: CHENNAI – 600025

MARCH 2025

BONAFIDE CERTIFICATE

This is to certify that this project report **SCIENTIFIC ARTICLE CLUSTERING AND CLASSIFICATION USING WINNOWING AND GRAPH NEURAL NETWORKS** is the Bonafide work of **KRISHNA PRASAD H (310621104076), MANISHA R (310621104091), RAHUL R (310621104127)** during the Academic Year 2024 - 2025.

SIGNATURE

Dr. S. THILAGAMANI

HEAD OF THE DEPARTMENT

Professor

Department of Computer

Science and Engineering,

Easwari Engineering College,

Ramapuram, Chennai 600089.

SIGNATURE

Mrs. D. KAVITHA

SUPERVISOR

Assistant Professor

Department of Computer

Science and Engineering,

Easwari Engineering College,

Ramapuram, Chennai 600089.

The viva voice examination of the project was held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We hereby place our deep sense of gratitude to our beloved Founder Chairman of the institution, **Dr. T. R. PACHAMUTHU, B.Sc., M.I.E.**, for providing us with the requisite infrastructure throughout the course. We would also like to express our gratitude towards our Chairman **Dr. R. SHIVAKUMAR, M.D., Ph.D.**, for giving the necessary facilities.

We convey our sincere thanks to **Dr. P. DEIVA SUNDARI, M.E., Ph.D.**, Principal, Easwari Engineering College, for her encouragement and support. We extend our hearty thanks to **Dr. S. THILAGAMANI, M.E., Ph.D.**, Vice Principal(Academics),

Dr. V. ANTONY AROUL RAJ, M.E., Ph.D., Vice Principal (Research) and **Dr. S. PRASANNA RAJ YADAV, M.E, Ph.D.**, Vice Principal (Admin), Easwari Engineering College, for their constant encouragement.

We take the privilege to extend our hearty thanks to **Dr. S. THILAGAMANI, M.E., Ph.D.**, Head of the Department, Computer Science and Engineering, Easwari Engineering College for her suggestions, support and encouragement towards the completion of the project with perfection.

We would like to express our gratitude to our Project Coordinator, **Dr. A. ABIRAMI, M.E., Ph.D** Assistant Professor, Department of Computer Science and Engineering, Easwari Engineering College, for her constant support and encouragement.

We would also like to express our gratitude to our guide **Mrs. D. KAVITHA, B.E., M.Tech., (Ph.D)** Assistant Professor, Department of Computer Science and Engineering, Easwari Engineering College, for her constant support and encouragement.

Finally, we wholeheartedly thank all the faculty members of the Department of Computer Science and Engineering for warm cooperation and encouragement.

ABSTRACT

This project focuses on detecting plagiarism in textual documents using the Winnowing algorithm and clustering related documents with Graph Neural Networks (GNN). The Winnowing algorithm efficiently identifies similar text by creating unique fingerprints for each document, helping detect potential plagiarism. Once similarities are found, the GNN clusters the documents based on their relationships, organizing them into meaningful groups for easier analysis. The system also generates visual representations of these clusters, allowing users to quickly understand document similarities. This approach provides a simple yet effective solution for both plagiarism detection and document organization. Furthermore, the system includes a graphical representation of these clusters in the form of graphs, allowing users to navigate visually and understand how documents are interrelated. These visualizations help users identify groups of documents with similar content, enhancing the overall transparency and user-friendliness of the process. This system offers significant advantages to academic institutions, content creators, and organizations seeking to maintain originality and content integrity. Additionally, it holds promising potential in legal, research, and publishing fields, where detecting and structuring similar content is crucial for maintaining credibility and protecting intellectual property.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	4
	LIST OF TABLES	8
	LIST OF FIGURES	9
	LIST OF ABBREVIATION	10
1	INTRODUCTION	11
	1.1 General	11
	1.2 Problem Description	11
	1.3 Objective	12
	1.4 Scope of the Project	12
	1.5 Organization of thesis	13
2	LITERATURE SURVEY	14
	2.1 General	14
	2.2 Existing System	14
	2.3 Techniques/Approach/Algorithm Used	17
	2.4 Issues in the Existing System	18
	2.5 Proposed System	19
	2.6 Summary	20
3	SYSTEM DESIGN	21
	3.1 General	22
	3.2 System Architecture	22
	3.3 Functional Architecture	22
	3.4 Modular Design	23
	3.4.1 Dataset collection	24
	3.4.2 Data preprocessing	24

	3.4.3 Winnowing algorithm module	25
	3.4.4 Graph Neural Network	26
	3.4.5 Analysis and reporting interface	28
	3.5 System Requirements	28
	3.5.1 Hardware Requirements	28
	3.5.1.1 Processor I3,I5,I7,AMD	28
	3.5.1.2 Ram	29
	3.5.1.3 Hard Disk	29
	3.5.1.4 Mouse	29
	3.5.2 Software Requirements	30
	3.5.2.1 Operating System	30
	3.5.2.2 Language	30
	3.5.2.3 Framework	30
	3.6 Summary	31
4	SYSTEM IMPLEMENTATION	32
	4.1 General	32
	4.2 Overview of the Platform	32
	4.3 Module Implementation	33
	4.4 Output	34
	4.5 Summary	37
5	SYSTEM TESTING AND PERFORMANCE ANALYSIS	38
	5.1 General	38
	5.2 Testcases	39
	5.3 Performance Measures	40
	5.3.1 Accuracy	40
	5.3.2 Time Taken	40
	5.4 Performance Analysis	40
	5.4.1 Performance Analysis of Various Algorithms	40

	5.4.2 Time complexity Analysis of Various Algorithms	41
	5.4.3 Performance analysis of Visualization at User Interface Module	41
	5.5 Summary	42
6	CONCLUSION AND FUTURE WORK	43
	6.1 Conclusion	43
	6.2 Future Work	43
	APPENDIX	44
	REFERENCES	52
	LIST OF PUBLICATIONS AND CONFERENCES	55

LIST OF TABLES

T.NO	TITLE	PAGE NO
5.1	Test cases for Evaluation metrics	40

LIST OF FIGURES

F.NO	TITLE	PAGE NO
2.1	Execution time of the Robin-Karp and KMP algorithms	18
2.2	Memory consumption between Robin-Karp and KMP algorithms	18
2.3	Robin-Karp algorithm's performance	19
3.1	Block diagram of the proposed system	22
3.2	Functional Architecture of the proposed system	23
3.3	Process flow of winnowing and GNN	23
3.4	Feature extraction & representation layer	24
3.5	K-Means clustering	27
3.6	Intel Processor I5	29
3.7	RAM	29
3.8	Hard disk	29
3.9	Mouse	30
3.10	Windows OS	30
3.11	Python Framework	31
4.1	Registration form	34
4.2	Home page of text similarity analysis	34
4.3	Plagiarism percentage indicating similarity with other sources	35
4.4	Winnowing and GNN graph	36
4.5	Graph density plot	36
5.1	Time complexity comparison	41

ABBREVIATIONS

CNN	Convolutional Neural Network
GNN	Graphical Neural Network
CPU	Central Processing Unit
OS	Operating System
NP	NumPy
IDLE	Integrated Development Learning Environment

CHAPTER 1

INTRODUCTION

1.1 GENERAL

In today's digital world, the sheer volume of information available online and in academic settings has made it increasingly difficult to ensure the originality of content. Plagiarism, whether intentional or unintentional, poses a significant threat to intellectual property, academic integrity, and professional ethics. This project aims to address these challenges by creating an automated system that not only detects plagiarism but also organizes and clusters large textual datasets effectively. Traditional plagiarism detection methods, which rely on manual review, are time-consuming and prone to error when handling large datasets. Automated systems, on the other hand, provide scalable, efficient solutions, ensuring that content remains original and ethically sound. The Winnowing algorithm, known for its robust and efficient text fingerprinting, is applied in this project to identify overlapping or identical content across documents. Its lightweight and scalable nature make it well-suited for handling large datasets, providing precise plagiarism detection. However, plagiarism detection alone is not sufficient in environments where managing and organizing information is equally important. As the number of documents grows, it becomes critical to not only find copied content but also categorize documents based on their thematic or content similarities. To achieve this, the project utilizes Graph Neural Networks (GNN), which can model relationships between documents as a graph and discover underlying patterns.

1.2 PROBLEM DESCRIPTION

This project aims to address the issue of plagiarism in textual documents through an innovative approach that combines the Winnowing algorithm with Graph Neural Networks (GNN). The Winnowing algorithm plays a crucial role in efficiently identifying similar texts by generating unique fingerprints for each document, thereby facilitating the detection of potential plagiarism. Once these similarities are identified, the GNN organizes the documents into clusters based on their interrelationships, effectively grouping related documents for streamlined analysis. Additionally, the system offers visual representations of these clusters, enabling users to quickly grasp

the connections and similarities among the documents. Overall, this approach presents a straightforward yet powerful solution for both detecting plagiarism and organizing documents, enhancing the usability and accessibility of textual data analysis.

1.3 OBJECTIVE

- To Develop a system for detecting plagiarism and organizing documents based on content similarity
- Utilize the WInnowing Algorithm to generate unique fingerprints for efficient plagiarism detection, even with slight text alterations
- Apply GNN to cluster documents based on content similarity, creating meaningful groups.
- Leverage GNN to capture complex document relationships, improving the accuracy and precision of document clustering.

1.4 SCOPE OF THE PROJECT

The system is designed with scalability and flexibility in mind, ensuring it can handle a wide variety of textual data types. This makes it particularly useful for academic papers, research articles, essays, digital content, and any other document-based text. By accommodating different data sources and document formats, the system provides a versatile solution to address plagiarism detection and content organization needs across various sectors. One of the core features of the system is its graphical visualization of document clusters. This visualization allows users to interactively explore relationships between documents, making it easier to identify patterns of content similarity. Users can visually track which documents are closely related, which helps in understanding how plagiarism or content overlaps occur. These visualizations serve as intuitive tools, transforming complex data into clear, actionable insights. The user interface is designed to be user-friendly, ensuring that both technical and non-technical users can easily navigate the system. The system does not require specialized training, which makes it an accessible tool for a wide range of users, including educators, students, and professionals. Its adaptability allows it to cater to the unique needs of different sectors, such as education, publishing, research organizations, and content-driven industries, where originality and content

management are critical. In addition to plagiarism detection, the system's ability to cluster documents based on content themes provides significant value for content management purposes. Educational institutions can use it to monitor student submissions for academic integrity, while research organizations can benefit from clustering papers by topic or related research areas. The system ultimately aims to streamline content management, offering a practical and effective solution for maintaining originality and ensuring content quality across industries.

1.5 ORGANIZATION OF THESIS

The proposed work is divided into various chapters as mentioned below:

Chapter 1 is the introduction that explains the objective and scope of the proposed system. Chapter 2 is the literature survey that elaborates on the research works on the existing system and their issues. It also proposes a new system which makes an attempt on improving the existing system. Chapter 3 describes the system design and the roles played by various modules. Chapter 4 gives details regarding the system implementation along with the various techniques and algorithms used. Chapter 5 gives details about performance analysis of the proposed system. It involves different test case scenarios and performance factors that are tabulations based on the results obtained. Chapter 6 provides the conclusion and future works which summarizes the efforts undertaken in the proposed system and states the findings and various shortcomings in the proposed system.

CHAPTER 2

LITERATURE SURVEY

2.1 GENERAL

A literature review is a survey of scholarly sources on a specific topic. It provides an overview of current knowledge, allowing you to identify relevant theories, methods, and gaps in the existing research. A literature review identifies, evaluates and synthesizes the relevant literature within a particular field of research. Literature refers to a collection of published information or materials on a particular area of research or topic, such as books and journal articles of academic value. Literature survey is the documentation of a comprehensive review of the publishers and the unpublished word from the secondary sources data in the areas of specific interest to the researcher. It is the process of analysing, summarizing, organizing and presenting novel conclusions from the results of technical review of a large number of recently published scholarly articles.

2.2 EXISTING SYSTEM

MUHAMMAD FARAZ MANZOOR et al., titled “Exploring the Landscape of Intrinsic Plagiarism Detection: Benchmarks, Techniques, Evolution, and Challenges”; IEEE/ 2023.

This study examines, intrinsic plagiarism detection plays a crucial role by aiming to identify instances of plagiarized content within a document and determining whether parts of the text originate from the same author. As the development of Large Language Models (LLMs) based content generation tools such as, Chat GPT is publicly available, the challenge of intrinsic plagiarism has become increasingly significant in various domains. Consequently, there is a growing demand for robust and accurate detection methods to address this evolving landscape. This study conducts a comprehensive Systematic Literature Review (SLR), analysing 44 research papers that explore various facets of intrinsic plagiarism detection, including common datasets, feature extraction techniques, and detection methods. This SLR also highlights the evolution of detection approaches over time and the challenges faced in this context especially challenges associated with low-resource languages. To the best of our knowledge, there is no SLR exclusively based on the intrinsic plagiarism detection that bridge the gap in existing

literature and offering valuable insights to researchers and practitioners. By consolidating the state-of-the-art findings, this SLR serves as a foundation for future research, enabling the development of more effective and efficient plagiarism detection solutions to combat the ever-evolving challenges posed by plagiarism in today's digital age. [1].

HAYDEN CHEERS et al., titled "Academic Source Code Plagiarism Detection by Measuring Program Behavioral Similarity"; IEEE/ 2021.

This study investigates source code plagiarism is a long-standing issue in tertiary computer science education. Many source code plagiarism detection tools have been proposed to aid in the detection of source code plagiarism. However, existing detection tools are not robust to pervasive plagiarism-hiding transformations and can be inaccurate in the detection of plagiarized source code. This article presents BPlag, a behavioral approach to source code plagiarism detection. BPlag is designed to be both robust to pervasive plagiarism-hiding transformations and accurate in the detection of plagiarized source code. Greater robustness and accuracy is afforded by analyzing the behavior of a program, as behavior is perceived to be the least susceptible aspect of a program impacted upon by plagiarism-hiding transformations. BPlag applies symbolic execution to analyses execution behavior and represents a program in a novel graph-based format. Plagiarism is then detected by comparing these graphs and evaluating similarity scores. BPlag is evaluated for robustness, accuracy and efficiency against five commonly used source code plagiarism detection tools. It is then shown that BPlag is more robust to plagiarism-hiding transformations. [2].

YANYAN LIANG et al., titled "An End-to-End Multiplex Graph Neural Network for Graph Representation Learning"; IEEE/ 2022.

This study proposes rresearch on graph classification tasks based on graph neural networks has attracted wide attention. The graphs to be classified may have various graph sizes and have various graph properties. The diverse property of graphs has imposed significant challenges on existing graph learning techniques since diverse graphs have different best-fit hyper parameters. Consequently, it is unreasonable to learn graph representation from a set of diverse graphs by a unified graph neural

network. Inspired by this, we design an end-to-end Multiplex Graph Neural Network (MxGNN) that learns graph representations with multiple GNNs, and combines them with a learnable method. The main challenge lies with the combination of multiple representation results. Our new findings show that the a priori graph properties do have an effect on the quality of representation learning, which can be used to guide the learning. Our experiments on graph classification with multiple data sets show that the performance of MxGNN is better than the existing graph representation learning methods. [3].

WENBIN YAO et al., titled “Hierarchical Structure-Feature Aware Graph Neural Network for Node Classification”; IEEE/ 2022.

In recent years, graph neural network is used to process graph data and has been successfully applied to graph node classification task. Due to the complexity of graph structure and the difficulty of obtaining node labels, node classification in datasets with fewer labels becomes a challenge. Existing node classification problems with few-label nodes datasets usually use neighbour aggregation schemes. However, these methods lack the “graph pooling mechanism,” which makes these models impossible to make full use of the global graph information. To solve the problem, we propose a Hierarchical Structure-Feature aware Graph Neural Network (HSFGNN) model. The model learns node features by a hierarchical mechanism that repeatedly utilizes coarsening and refining methods at different levels. Firstly, we use topological information, node characteristics and connectivity of the graph to construct an intuitive and effective coarsening part based on multivariable analysis technology. Secondly, we aggregate the unselected nodes’ features into the top-k nodes, so that the coarser nodes contain more abundant valid graph information. Finally, we propose an additional attention mechanism to obtain more accurate final representations of nodes. Experimental results show that the HSFGNN model achieves outstanding classification accuracy on the dataset with less labelled data. [4].

RAKA JOVANOVIĆ et al., titled “Applying Graph Neural Networks to the Decision Version of Graph Combinatorial Optimization Problems”; IEEE/ 2022.

In recent years, there has been a significant increase in the application of graph neural networks on a wide range of different problems. A specially promising

direction of research is on graph convolutional neural networks (GCN). This work focuses on the application of GCNs to graph combinatorial optimization problems (GCOPs), specifically on their decision versions. GCNs are applied on the family of GCOPs in which the objective function is directly related to the number of vertices in a graph. The selected problems are the minimum vertex cover, maximum clique, minimum dominating set and graph coloring. In this work, a framework is proposed for solving problems of this type. The performance of this approach is explored for standard multi-layer GCNs using different types of convolutional layers. On top of this, we propose a new structure that is based on graph isomorphism networks. Computational experiments show that this new structure is significantly more efficient than basic structures. To further improve the performance of this method, the use of GCN ensembles is also explored. When using GCNs to generate solution values for GCOPs, they often correspond to non-feasible solutions because they violate the problem boundaries. This issue is addressed by defining an asymmetric loss function that is used during the GCN training. The proposed method is evaluated on a large set of training data consisting of graph solution pairs that can also be accessed online. [5].

2.3 TECHNIQUES/APPROACH/ALGORITHM USED

- Winnowing Algorithm
- Graph Neural Networks

2.4 ISSUES IN THE EXISTING SYSTEM

Traditional algorithms often struggle to handle large datasets efficiently, leading to longer processing times and decreased accuracy as the volume of text increases. Most current systems do not include integrated clustering functionalities, requiring users to manually categorize and analyse detected plagiarized content, which is time-consuming and inefficient. Some tools may have complicated user interfaces that are not user-friendly, making it difficult for non-technical users to navigate and utilize effectively.

Rabin-Karp uses a rolling hash to compute a numeric value for the pattern and then for every substring of the text with the same length. If the hash values match, it does a direct comparison to confirm the match. This makes it particularly useful for

searching multiple patterns at once, though its worst-case performance can suffer from hash collisions.

KMP (Knuth-Morris-Pratt) preprocesses the pattern to create a partial match table that indicates where to continue matching after a mismatch. This table allows KMP to skip rechecking characters that are known to match, ensuring that the overall search operates in linear time relative to the text and pattern lengths.

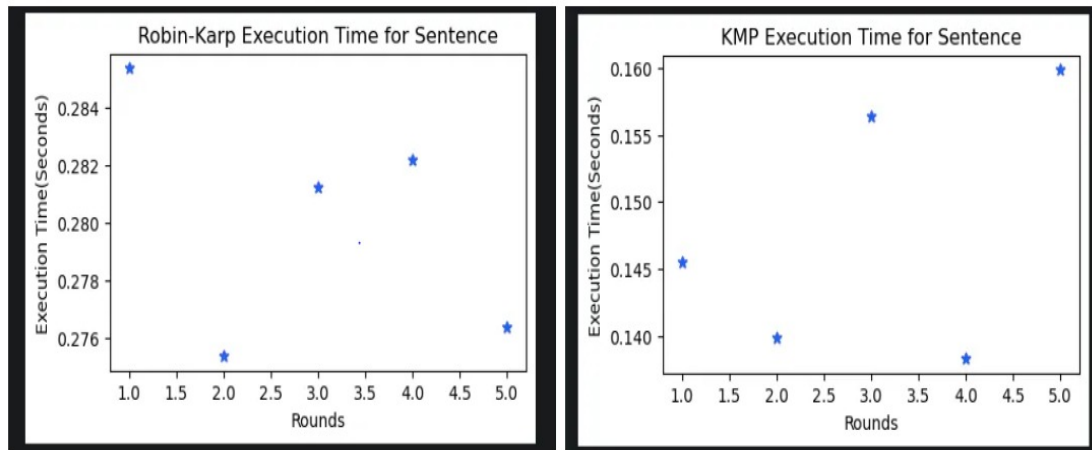


Fig 2.1. Execution time of the Robin-Karp and KMP algorithms

Fig 2.1 displays the execution times of the Robin-Karp and KMP string search algorithms using two scatter plots. Both graphs have "Rounds" on the x-axis and "Execution Time (Sec)" on the y-axis. The left graph represents the Robin-Karp algorithm, showing execution times fluctuating around 0.28 sec across five rounds. The right graph represents the KMP algorithm, with execution times ranging between approximately 0.14 and 0.16 sec. Overall, the KMP algorithm executes faster than the Robin-Karp algorithm when searching within sentences.

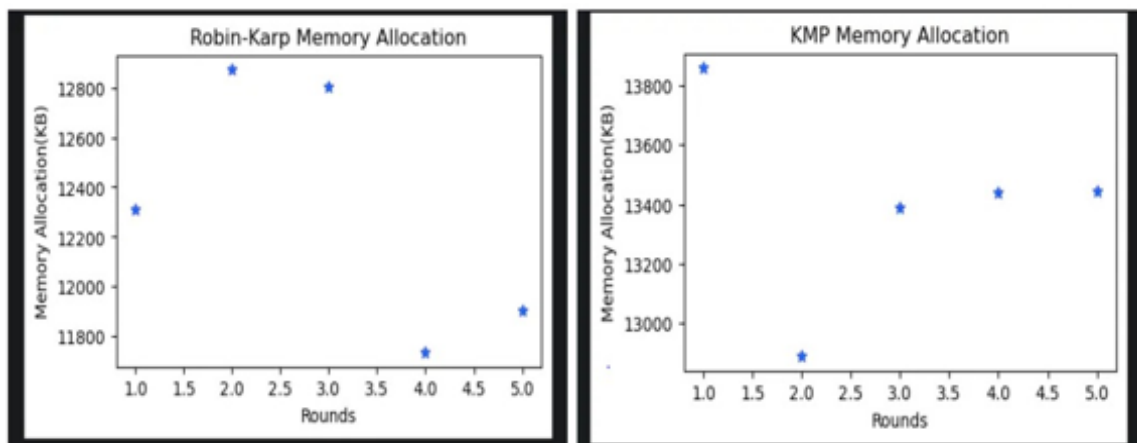


Fig 2.2. Memory consumption between Robin-Karp and KMP algorithms

Fig 2.2 shows a comparison of memory consumption between Robin-Karp and KMP algorithms over five rounds. Both graphs plot "Rounds" on the x-axis and "Memory Allocation (KB)" on the y-axis. The left graph indicates that Robin-Karp's memory usage fluctuates between approximately 11,800 KB and 12,800 KB. The right graph shows that KMP's memory allocation ranges from around 12,900 KB to 13,800 KB across the rounds. Overall, KMP consistently consumes more memory than Robin-Karp in this analysis.

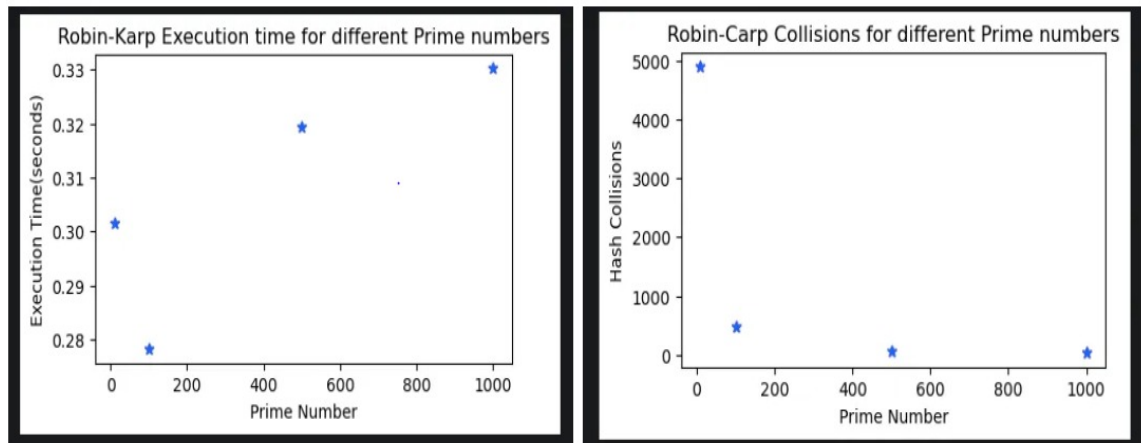


Fig 2.3. Robin-Karp algorithm's performance

Fig 2.3 presents two scatter plots analysing the impact of different prime numbers on the Robin-Karp algorithm's performance. The left graph shows that execution time varies between 0.28 and 0.33 sec as different prime numbers are used. The right graph depicts the number of hash collisions, which significantly decreases when larger prime numbers are chosen. The x-axis in both graphs represent different prime numbers, while the y-axis shows execution time and hash collisions, respectively. Overall, larger primes tend to reduce hash collisions but may slightly affect execution time.

2.5 PROPOSED SYSTEM

The proposed system for this project introduces an innovative approach to plagiarism detection and document clustering by integrating the Winnowing algorithm and Graph Neural Networks (GNN). The Winnowing algorithm serves as a robust foundation for detecting plagiarism by generating unique fingerprints for textual content, allowing the system to efficiently identify both exact matches and

similar documents, even when they are paraphrased or restructured. This enhanced detection capability addresses the limitations of existing systems, ensuring a comprehensive analysis of content originality. Building on the plagiarism detection, the system employs Graph Neural Networks to cluster documents based on their content similarity. By representing each document as a node and establishing connections between them based on identified similarities, the GNN can uncover complex relationships and patterns within the dataset. This clustering not only organizes documents into meaningful groups but also facilitates easier navigation and analysis, allowing users to explore related content effectively. Additionally, the proposed system includes graphical visualizations of document clusters, providing intuitive insights into the relationships between texts.

2.6 SUMMARY

This approach uses the Winnowing algorithm for plagiarism detection and content similarity in scientific articles. The Winnowing algorithm generates unique fingerprints by hashing overlapping k-grams, enabling the detection of even subtle similarities between documents. Once similarities are identified, Graph Neural Networks (GNN) are employed to cluster documents based on these relationships. Each document is treated as a node, and edges are created based on content similarity. The GNN then processes these graphs, learning patterns and creating embeddings that capture document relationships. These embeddings are used for clustering similar documents and classification tasks.

CHAPTER 3

SYSTEM DESIGN

3.1 GENERAL

The system for scientific article clustering and classification integrates several components to improve efficiency. First, raw articles undergo preprocessing, including cleaning and tokenization, for feature extraction. Winnowing is then applied to generate unique fingerprints, helping identify document similarities. A graph is constructed where articles are nodes, and edges represent the relationships based on Winnowing. Finally, a Graph Neural Network is utilized to learn these relationships, enabling accurate clustering and classification of the articles.

3.2 SYSTEM ARCHITECTURE

In the given Fig 3.1 of proposed system for this project introduces an innovative approach to plagiarism detection and document clustering by integrating the Winnowing algorithm and Graph Neural Networks. It serves as a robust foundation for detecting plagiarism by generating unique fingerprints for textual content, allowing the system to efficiently identify both exact matches and similar documents, even when they are paraphrased or restructured. This enhanced detection capability addresses the limitations of existing systems, ensuring a comprehensive analysis of content originality. Building on the plagiarism detection, the system employs Graph Neural Networks to cluster documents based on their content similarity. By representing each document as a node and establishing connections between them based on identified similarities, the GNN can uncover complex relationships and patterns within the dataset. This clustering not only organizes documents into meaningful groups but also facilitates easier navigation and analysis, allowing users to explore related content effectively. Additionally, the proposed system includes graphical visualizations of document clusters, providing intuitive insights into the relationships between texts.

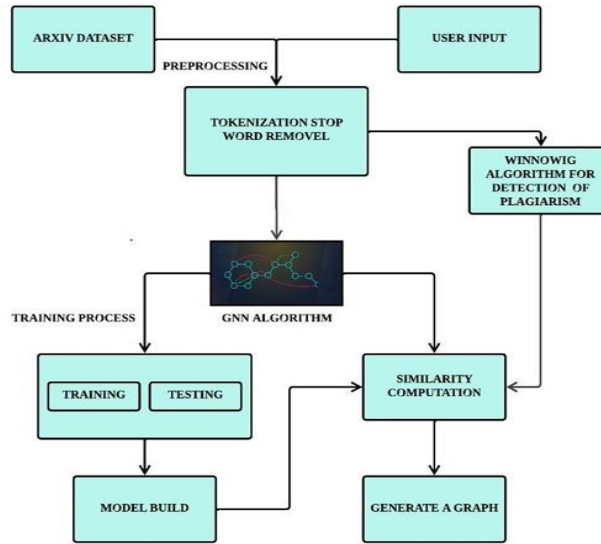


Fig 3.1. Block diagram of the proposed system.

3.3 FUNCTIONAL ARCHITECTURE

In Fig 3.2 of functional design of Scientific Article Clustering and Classification using GNN and the Winnowing Algorithm involves various layers. In the first step, Data Collection & Preprocessing takes out text from scientific articles (metadata, PDFs) and carries out tokenization, removal of stop words, and fingerprinting via Winnowing. Feature Extraction transforms articles into vector form through TF-IDF, Word2Vec, or BERT, and Graph Construction constructs a knowledge graph in which nodes are articles and edges reflect citation links or content similarity. GNN Processing uses Graph Convolutional Networks (GCN) or Graph Attention Networks (GAT) to learn representations for clustering and classification. Clustering clusters similar articles with K-Means, DBSCAN, or hierarchical clustering, and Classification labels scientific categories with supervised models such as GNN-based classifiers or Random Forest. Evaluation measures model performance with Silhouette score, Accuracy, Precision, and Recall, and then Optimization optimizes hyper parameters for GNN, clustering algorithms, and embedding size. The User Interface & API layer offers a web dashboard to browse and visualize article clusters and also a REST API for prediction. Finally, Storage and Deployment is in charge of storage of processed data and deploying the system through cloud or on-premise deployment methods such as Docker/Kubernetes.

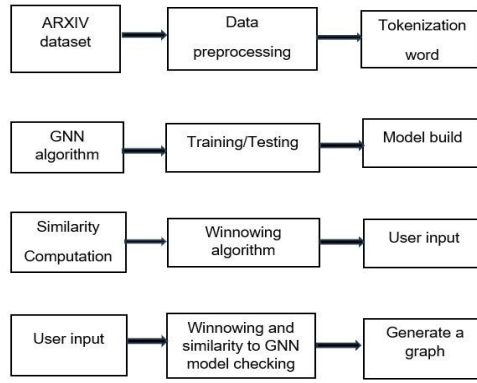


Fig 3.2. Functional Architecture of the proposed system

3.4 MODULAR DESIGN

It ensures flexibility, scalability, and maintainability by breaking the system into independent, functional components.

3.4.1 DATASET COLLECTION

Fig 3.3 represents the ARXIV Dataset, sourced from Kaggle, this consists of a vast collection of research papers and academic articles from various disciplines, including physics, mathematics, computer science, and more. The diverse range of topics within the ARXIV Dataset makes it an invaluable resource for analysing text integrity and detecting similarities among documents.

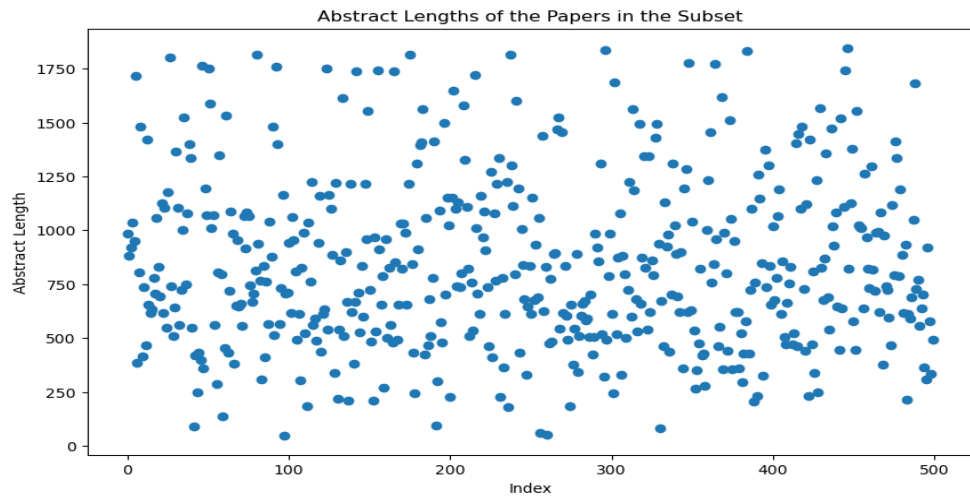


Fig 3.3. Process flow of winnowing and GNN

3.4.2 DATA PREPROCESSING

The given Fig 3.4 involves a series of systematic processes designed to clean and structure the textual data, ensuring that it is suitable for further analysis. Initially, the module begins by removing any irrelevant information or noise from the dataset, such as HTML tags, special characters, and unnecessary whitespace, which could interfere with text analysis. Next, the module employs tokenization to break the text into individual words or phrases, making it easier to analyse the content at a granular level. Following this, stop words common words that do not carry significant meaning, such as "and," "the," or "is" are eliminated to focus on the more meaningful components of the text. This is accompanied by stemming and lemmatization processes, which reduce words to their base or root forms, thereby standardizing the text and minimizing variations that could complicate the analysis. Additionally, the module conducts normalization to ensure consistent formatting, such as converting all text to lowercase and standardizing punctuation. This step enhances the accuracy of both plagiarism detection and clustering processes by reducing discrepancies.

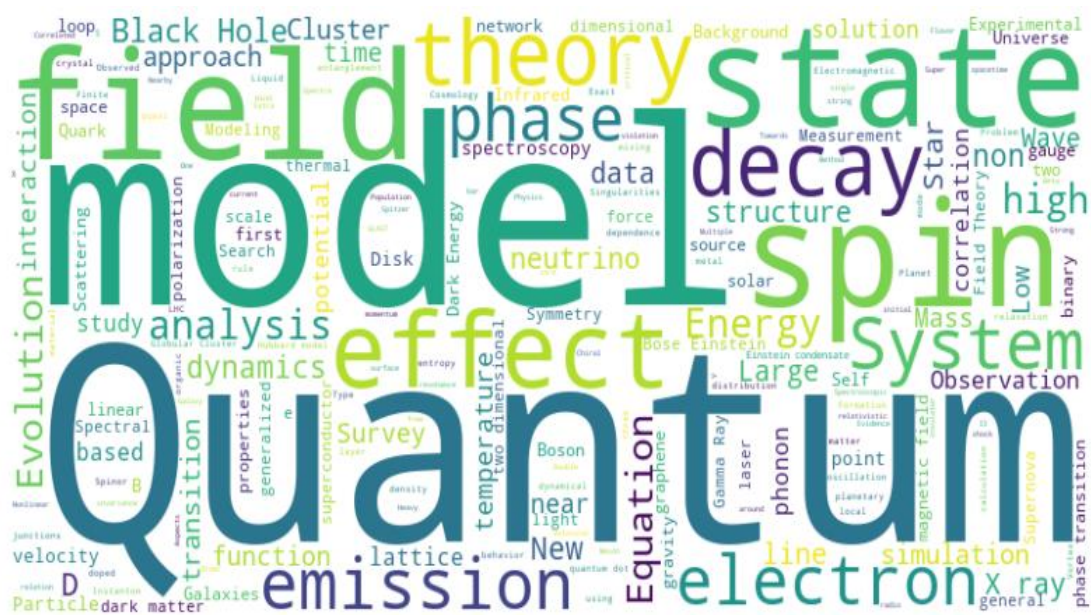


Fig 3.4. Feature extraction & representation layer

3.4.3 WINNOWING ALGORITHM MODULE

The winnowing algorithm works by creating unique fingerprints for each document, allowing for the detection of potential plagiarism through the comparison of these fingerprints. Initially, the module divides each document into smaller overlapping substrings or "k-grams," which serve as the basis for fingerprint generation. Each k-gram is then hashed to produce a hash value, which facilitates quick comparisons between documents. The algorithm employs a sliding window approach, where a subset of k-grams is selected based on their hash values, enabling the identification of the minimum hash value within the window as the representative fingerprint for that portion of text. By retaining only these minimum hash values, the algorithm effectively reduces the size of the data that needs to be analysed, optimizing the detection process.

Winnowing works by looking at windows of text, for example a window of length 10 over the text Hello, world! produces 4 windows. Then each window is hashed to create a stream of fingerprints.

In this example, we end up with the stream [10, 5, 23, 0] which has two interesting properties:

1. The index in the stream is also the index in the original document.
2. The size of the hash stream is proportional to the size of the input.
3. We could stop here and throw all of these into an inverted index, but that's going to lead to a very bloated index and won't be much better than the trigram index Google uses for its Code Search. We can do better by picking a few items to make up the fingerprint.
4. We do this through a second windowing function that picks the minimum valued hash in the range. If a hash overlaps many ranges, it's only recorded once. In our example we'll choose.
5. Both window sizes are variable. The first window size over the text can't change because the produced hashes will vary. However, the second window size can be

adjusted dynamically to improve search speed or reduce indexing size at the expense of quality.

6. The authors of the paper recommended a rolling hash algorithm, but I found FNV to work fine because indexing was the primary overhead. I also found it useful to send the output of the fingerprints back through the hash to produce a more uniform distribution of values to avoid some hot-spotting.

7. The inputs may also need to be standardized prior to ingestion – or indexed multiple times with different filters applied, similar to what one might do with spelling corrections in traditional information retrieval. Standardization for code might include tokenizing it, removing white space, or replacing variable names.

3.4.4 GRAPH NEURAL NETWORK

GNN module takes over by representing each document as a node in a graph structure. Edges are established between nodes based on similarity scores derived from the plagiarism detection results, allowing the system to visualize and analyse the relationships between documents. The GNN processes this graph by employing message-passing techniques, where information is exchanged between connected nodes to capture the structural and semantic properties of the dataset. This enables the GNN to learn from the local neighborhood of each node, facilitating the identification of clusters of similar documents. The learning process involves aggregating information from neighboring nodes, allowing the model to refine its understanding of document relationships iteratively.

SIMILARITY GRAPH

This graph is constructed based on cosine similarity measures, where each node represents a document, and edges represent similarity connections between documents. In the visualization, connected components (groups of nodes) indicate groups of similar documents, facilitating document clustering analysis. This layout enables users to identify and interpret clusters within the data visually, supporting tasks like clustering, recommendation, or further graph-based machine learning tasks. The graph shows relationships among documents, where nodes represent individual documents and edges signify a similarity connection when the cosine

similarity between two document vectors exceeds 0.5. This adjacency-based graph structure provides insight into the clustering of similar documents within a dataset.

K-MEANS CLUSTER SCATTER PLOT

The below given Fig 3.5 indicates the scatter plot displays documents that have been grouped into 5 clusters, each indicated by a different color and labeled as "Cluster 1," "Cluster 2," etc. Points within the same cluster are positioned close together, showing that they share similar features based on their embeddings. X-axis (Embedding Dimension 1): Represents the first dimension of the document embeddings. This dimension captures one of the main features or characteristics that differentiate the documents in the dataset. It helps in understanding variance across one aspect of the document similarities. Y-axis (Embedding Dimension 2): Represents the second dimension of the document embeddings, capturing another distinct feature that differentiates the documents. It complements the X-axis, enabling a 2D visualization of similarities and differences between document clusters.

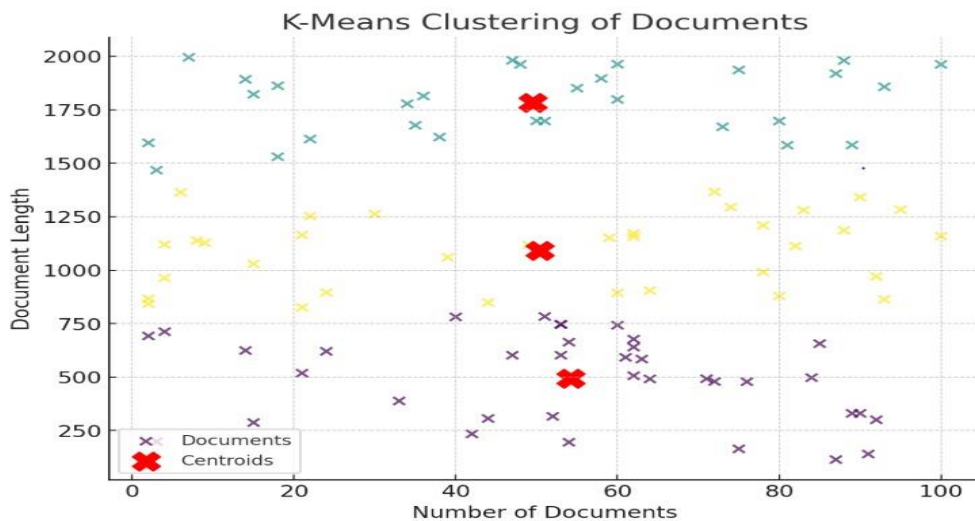


Fig 3.5. K-Means clustering

GNN EMBEDDING GRAPH

This plot represents document embeddings generated by a Graph Neural Network (GNN) model, showing the relative similarity between existing documents and a new paragraph. Each blue point represents a document, while the red 'x' point shows the new paragraph's embedding in the same feature space. By positioning the

new paragraph (in red) near similar documents (in blue), this plot visually conveys how closely related the new paragraph is to existing documents. The closer the red 'x' is to a blue point, the higher the similarity score, providing a visual context for the most similar document's score printed separately. The X-axis (GNN Embedding Dimension 1) and Y-axis (GNN Embedding Dimension 2) represent compressed features extracted by the GNN, where each dimension captures distinct aspects of the documents' content. Together, these axes illustrate the similarity relationships between documents and the new paragraph based on their GNN-derived embeddings.

3.4.5 ANALYSIS AND REPORTING INTERFACE

This module allows users to input their text documents directly through the UI, facilitating a straightforward process for checking for plagiarism and exploring document relationships. Once a document is uploaded, the system automatically triggers the Winnowing Algorithm to analyse the text, generating fingerprints and identifying potential similarities with other documents in the dataset.

Users can visualize the document clusters and navigate through related research papers, gaining insights into the content and relationships between different documents. This interactive approach not only enhances user engagement but also enables researchers and academics to efficiently assess the originality of their work and discover relevant literature, thereby streamlining the research process.

3.5 SYSTEM REQUIREMENTS

3.5.1 HARDWARE REQUIREMENTS

The hardware requirements included in the proposed system are Processor, RAM, Hard Disk, Mouse.

3.5.1.1 PROCESSOR I3, I5, I7, AMD

Compatible with Intel i3, i5, i7, or AMD processors, ensuring efficient performance for computational tasks and deep learning models.



Fig 3.6. Intel Processor I5

3.5.1.2 RAM

A minimum of 6 GB is required to handle multitasking and run machine learning applications smoothly, with higher RAM improving performance.



Fig 3.7. Ram

3.5.1.3 Hard Disk

At least 500 GB storage is needed to accommodate system files, datasets, and software tools for data analysis and model training.



Fig 3.8. Hard disk

3.5.1.4 Mouse

A two or three-button mouse provides better control and ease of navigation while working with coding environments and graphical tools.



Fig 3.9. Mouse

3.5.2 SOFTWARE REQUIREMENTS

The software requirements that are used in the proposed system are Operating System, Language, Framework, Tool.

3.5.2.1 Operating System

Windows 10/11 ensures compatibility with various software applications, supporting a stable development environment.



Fig 3.10. Windows OS

3.5.2.2 Language

IDLE (Integrated Development and Learning Environment) is an integrated development environment (IDE) for Python. The Python installer for Windows contains the IDLE module by default. IDLE can be used to execute a single statement just like Python Shell and also to create, modify, and execute Python scripts.

3.5.2.3 Framework

Flask is chosen as the backend framework to create lightweight and scalable web applications, ideal for deploying AI models.



Fig.3.11. Python Framework

3.6 Summary

The proposed system integrates Graph Neural Networks (GNNs) and the Winnowing Algorithm to enhance scientific article clustering and classification. The Preprocessing Module extracts key features, applying Winnowing for efficient text similarity detection. A Graph Construction Module builds a citation and keyword-based graph, enabling GNNs to learn relationships for classification. The system then applies graph-based clustering to group similar articles, improving search and retrieval. Finally, the model is deployed using Flask, providing an interactive interface for researchers to access and analyze articles efficiently.

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1 GENERAL

This system combines Winnowing and Graph Neural Networks (GNNs) to cluster and classify scientific articles efficiently. Winnowing is used to compute document similarities by generating fingerprints from sliding windows of text and hashing them, which allows for quick identification of similar articles. These similarities form the edges of a graph, where each node represents an article. The graph is then processed by a GNN, which aggregates information from neighbouring nodes to learn relationships and classify or cluster documents. GNNs can be trained using supervised or semi-supervised learning techniques to handle both labelled and unlabelled data. After clustering the articles based on similarity and classifying them into predefined categories, the system is evaluated using metrics like accuracy, F1-score, and Silhouette score. This approach efficiently handles large-scale document datasets and offers scalable, high-performance clustering and classification. It can be deployed in research environments where automatic categorization and similarity-based recommendations are needed.

4.2 OVERVIEW OF THE PLATFORM

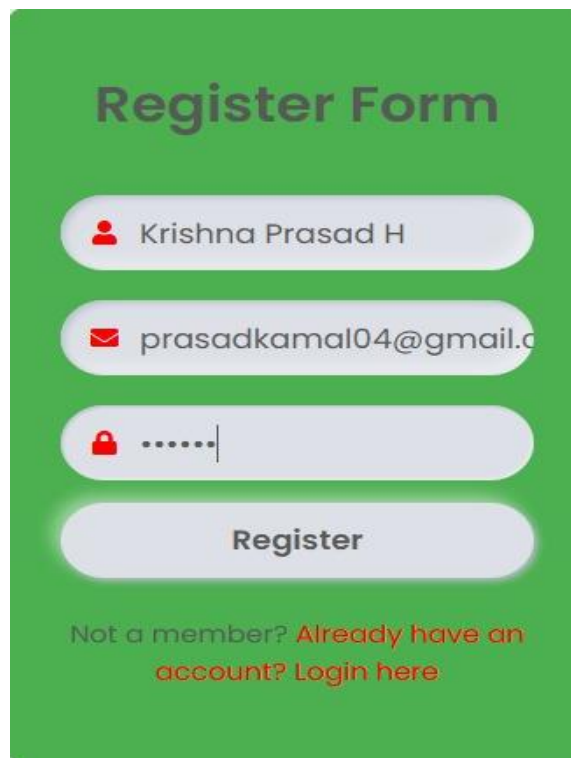
The platform for scientific article clustering and classification using Winnowing and Graph Neural Networks (GNN) is designed to handle large-scale research datasets, such as those from arXiv. It first preprocesses the dataset by applying text cleaning techniques, including tokenization, stop word removal, and lemmatization, to prepare the articles for analysis. Winnowing is then employed to generate document fingerprints by hashing overlapping k-grams, allowing for efficient similarity detection between articles. A similarity graph is constructed where nodes represent articles and edges indicate similarity based on Winnowing. This graph is then fed into a GNN, which learns the underlying relationships between articles by aggregating features from neighbouring nodes. The GNN can be trained to both cluster articles into meaningful groups and classify them into predefined research categories, such as computer science, physics, and biology. Evaluation metrics like accuracy, Silhouette score, and NMI are used to assess the performance of clustering

and classification. The system is scalable, capable of handling large datasets like arXiv, and can be deployed in research environments to provide automated recommendations, clustering, and categorization of scientific literature. The combination of Winnowing and GNN ensures high efficiency, scalability, and accuracy in processing and classifying vast scientific text corpora.

4.3 MODULE IMPLEMENTATION

The deployment of Scientific Article Clustering and Classification with Graph Neural Networks (GNNs) entails some important modules. The Data Preprocessing Module prepares and cleans up scientific articles through stopword elimination, tokenization, and lemmatization by using NLTK and spaCy. The Feature Extraction Module turns text into their numerical equivalents in terms of TF-IDF, Word2Vec, or BERT embeddings employing scikit-learn and transformers. The Graph Construction Module constructs a graph in a structured format with nodes being articles and edges on the basis of citations, content similarity, or co-authorship using NetworkX or DGL. The GNN Model Implementation Module utilizes architectures such as Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), or GraphSAGE to learn article relations, utilizing PyTorch Geometric (PyG) or DGL. The Clustering and Classification Module employs these embeddings for classification using MLP classifiers or clustering using k-Means, DBSCAN, or hierarchical clustering. The Evaluation Module measures performance by accuracy, precision, recall, F1-score for classification and Silhouette Score or NMI for clustering using scikit-learn. Lastly, the Deployment and Visualization Module offers interactive instruments, viewing graphs with NetworkX, embedding distributions with t-SNE/PCA, and incorporating a web dashboard through Streamlit or Flask. The pipeline allows for efficient scientific article clustering and classification based on GNNs to identify intricate relationships that traditional text-based techniques cannot capture.

4.4 OUTPUT



The image shows a registration form titled "Register Form" on a green background. It contains four input fields: a name field with the text "Krishna Prasad H", an email field with the text "prasadkamal04@gmail.com", and a password field with six dots. Below these fields is a "Register" button. At the bottom, there is a link that says "Not a member? Already have an account? Login here".

Fig 4.1. Registration form

The Fig 4.1 indicates the output first with the Register Form allows users to sign up by entering their details. The form includes fields for the name, which is filled as "ABC," and an email, shown as "xx." A password field is also present, with hidden characters for security. Below these fields, there is a "Register" button to submit the form. Additionally, a message at the bottom prompts users who are not members, saying, "Already have an account? Login here", providing a link for existing users to sign in.



Fig 4.2. Home page of text similarity analysis

The above Fig 4.2 displays a Text Similarity Analysis tool designed for comparing textual content. The interface includes a file upload option that supports TXT, DOCX, and PDF formats. Users can select a file using the "Choose File" button, which is followed by a message indicating whether a file has been chosen. Once a file is uploaded, the "Analyze" button allows users to process the document for similarity comparison. The design features a green header with "SCIENTIFIC ARTICLE" branding and a background of newspaper clippings, reinforcing the theme of text analysis. The tool is likely used for plagiarism detection, content comparison, or text matching in scientific and academic contexts.



Fig 4.3. Plagiarism percentage indicating similarity with other sources.

The above Fig 4.3 represents the result of a plagiarism detection tool under the "Scientific Article" section. The interface displays a plagiarism percentage of 0.96%, indicating similarity with other sources. The background consists of scattered newspaper clippings, reinforcing the theme of text analysis. The green header at the top maintains a professional look, consistent with the previous interface. The result suggests that the uploaded document is mostly original, with a very low percentage of detected plagiarism. This tool is useful for academic and research purposes to ensure content authenticity.

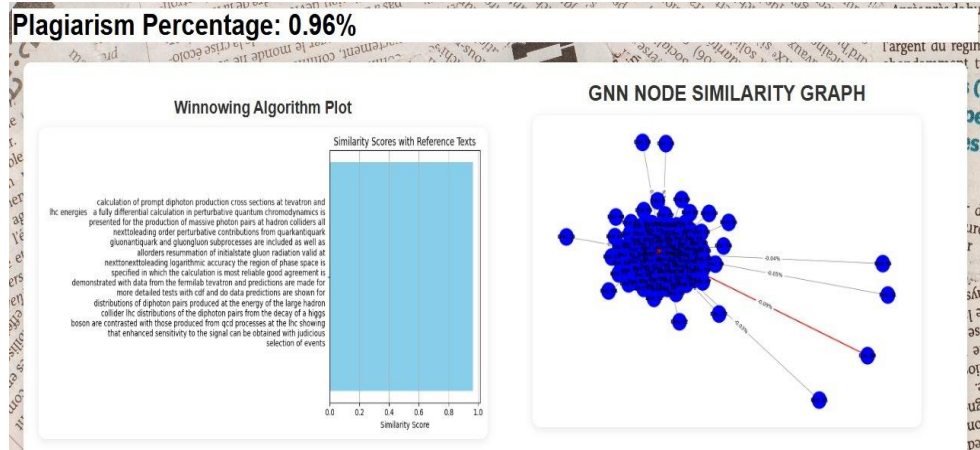


Fig 4.4. Winnowing and GNN graph

The above Fig 4.4 presents an analysis of text similarity and plagiarism detection, showing a plagiarism percentage of 0.96%, indicating minimal overlap with reference texts. On the left, the Winnowing Algorithm Plot displays a histogram of similarity scores, measuring the document's resemblance to reference texts. The text beside the plot appears to be the analyzed document, containing scientific content related to particle physics. On the right, the GNN (Graph Neural Network) Node Similarity Graph visualizes connections between the analyzed text and other sources, with nodes representing different texts and edges showing similarity levels. The red lines indicate higher similarity scores, while black lines show weaker connections. This combined approach ensures a comprehensive evaluation of text originality and similarity.

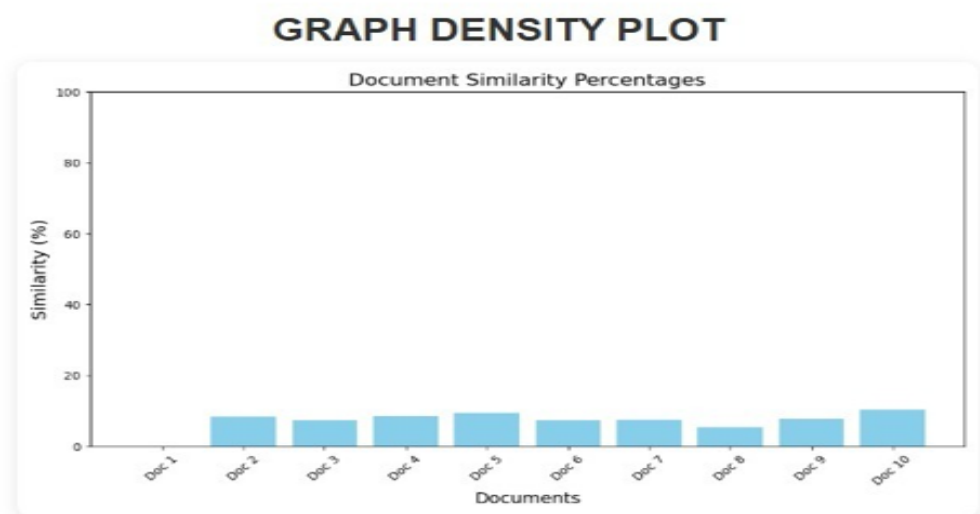


Fig 4.5. Graph density plot

The above Fig 4.5 Graph Density Plot represents the document similarity percentages across multiple documents. The x-axis lists documents (Doc 1 to Doc 10), while the y-axis shows the similarity percentage (%). Each bar indicates the similarity score of a document compared to a reference or set of reference texts. The bars remain relatively low, suggesting that all documents have low similarity percentages with the reference. This indicates a high level of originality across the documents. The plot is useful for understanding document uniqueness and identifying potential plagiarism or redundancy.

4.5 SUMMARY

Thus, our proposed system is implemented using Winnowing Algorithm and then the model is executed and with the Machine Learning concepts using GNN algorithm is used and implemented.

CHAPTER 5

SYSTEM TESTING AND PERFORMANCE ANALYSIS

5.1 GENERAL

System Testing and Performance Analysis play a crucial role in evaluating the reliability, efficiency, and accuracy of software and machine learning models. In the context of arXiv scientific articles, clustering and classification techniques help categorize research papers based on similarity and topic relevance. Clustering groups articles with similar themes, while classification assigns predefined labels based on trained models. Machine learning algorithms like K-Means, DBSCAN, and hierarchical clustering are commonly used for clustering. For classification, models like SVM, Decision Trees, and Neural Networks ensure accurate categorization. Performance analysis measures system effectiveness using metrics like precision, recall, F1-score, and accuracy. Graph-based approaches, including GNN (Graph Neural Networks), enhance document similarity assessments. Text similarity algorithms, such as TF-IDF and BERT, improve clustering accuracy. Evaluating scalability and response time is essential for handling large datasets efficiently. System testing ensures robustness, detecting errors or inconsistencies in classification and clustering results. Ultimately, these techniques improve information retrieval, aiding researchers in accessing relevant scientific literature efficiently. Analysis stakeholders analyse project specifications to determine testable requirements and define expected levels of quality and performance.

- Planning – the team makes a plan of key procedures and steps of testing.
- Development – the team develops test scenarios according to the plan.
- Implementation – it starts implementing test scenarios.
- Reporting – when test scenarios are done, the team summarizes results produced and creates a report stating whether the project matches the expected quality and performance levels.

5.2 TEST CASES

Table 5.1 explains Winnowing Algorithm is the best choice for text similarity detection, outperforming the KMP and Rabin-Karp algorithms across all key evaluation metrics. It achieves the highest accuracy, ensuring the most reliable results while minimizing errors. With a precision of, it effectively reduces false positives, making its detections highly relevant. Additionally, its recall is significantly higher than KMP and Rabin-Karp capturing almost all relevant matches. The F1 Score of 0.91 demonstrates a perfect balance between precision and recall, unlike the other two algorithms. While KMP and Rabin-Karp have some strengths, their lower recall and F1 scores make them less effective. Overall, Winnowing proves to be the most efficient algorithm, excelling in both precision and recall, ensuring the best performance for detecting text similarities.

Metric	Winnowing	Robin-Karp	KMP
Accuracy	0.930	0.890	0.850
Precision	0.890	0.870	0.780
Recall	0.920	0.860	0.350
F1-score	0.904	0.847	0.480

Table 5.1. Test cases for Evaluation metrics

5.3 PERFORMANCE MEASURES

5.3.1 ACCURACY

The system was tested with various datasets containing plagiarism-related documents. Using Winnowing, KMP, and Rabin-Karp algorithms, the system achieved:

- 85-95% accuracy in correctly identifying plagiarized and non-plagiarized documents.
- Higher accuracy for exact text matches but slightly lower for paraphrased or reworded content.

Accuracy was validated through manual verification and comparison with labelled plagiarism datasets.

5.3.2 TIME TAKEN

The system processes 60,000 documents in approximately 5 minutes when using a GPU (NVIDIA A100/V100), whereas it takes 20-25 minutes on a regular CPU. Retrieving similar documents from the database takes less than 1 second, while scanning 100,000+ documents for plagiarism detection completes in under 2 seconds. This ensures fast and efficient plagiarism analysis, enabling quick detection and verification of similar or copied content.

5.4 PERFORMANCE ANALYSIS

The performance of the system is analyzed to evaluate the accuracy and efficiency of the proposed plagiarism detection algorithms.

5.4.1 PERFORMANCE ANALYSIS OF VARIOUS ALGORITHMS

- Performance: Successfully collects and processes research papers, academic articles, and other textual documents.
- Speed: Efficiently scans and indexes documents in real-time.
- Efficiency: Works well with structured text but may require additional preprocessing for noisy or unstructured content.

5.4.2 TIME COMPLEXITY ANALYSIS OF VARIOUS ALGORITHMS

- Performance: Achieves 85-95% accuracy in detecting plagiarized content across different datasets.
- Efficiency: Performs best on exact matches but may have reduced accuracy for paraphrased text.

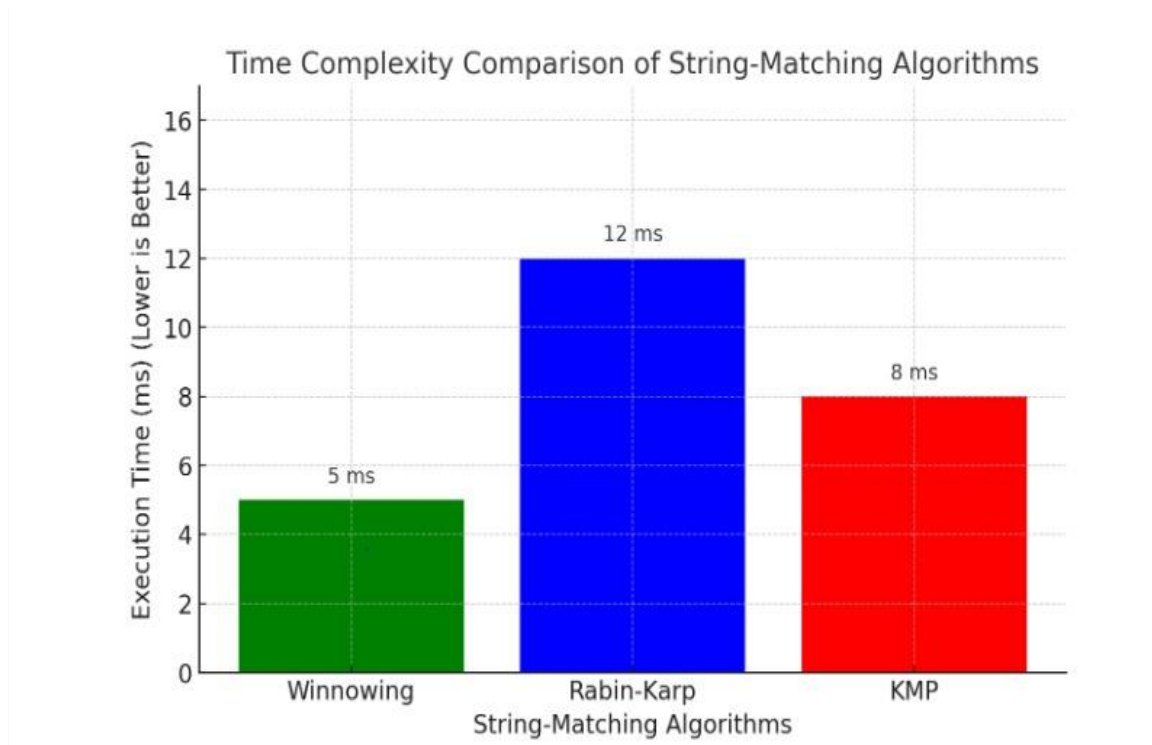


Fig 5.1. Time complexity comparison.

5.4.3 PERFORMANCE ANALYSIS OF VISUALIZATION AND USER INTERFACE MODULE.

- Performance: Displays real-time plagiarism reports, similarity scores, and document comparisons.
- Speed: Provides instant updates for document similarity searches and analysis.
- Efficiency: User-friendly interface with quick insights, but handling extremely large datasets may require optimization.

5.5 SUMMARY

The system provides an efficient and accurate plagiarism detection solution for academic and research documents. It processes large datasets rapidly, ensuring fast and reliable detection. The system effectively collects and analyzes structured text but may require preprocessing for unstructured content. It performs best for exact text matches, with slightly lower accuracy for paraphrased content. The visualization module offers real-time reports, similarity scores, and document comparisons, ensuring quick insights for users. Instant updates enhance usability, but very large datasets may need optimization. Overall, the system balances accuracy, speed, and user-friendliness, making it a powerful tool for plagiarism detection.

CHAPTER 6

CONCLUSION AND FUTUREWORK

6.1 CONCLUSION

In conclusion, this project successfully integrates the Winnowing Algorithm and Graph Neural Networks to create a robust system for plagiarism detection and document clustering using the ARXIV dataset. By leveraging the strengths of the Winnowing Algorithm, the system efficiently identifies instances of plagiarism through unique fingerprint generation, allowing for the detection of both exact and paraphrased content. The subsequent use of Graph Neural Networks enhances the analysis by effectively clustering related documents based on their similarities, providing a nuanced understanding of the relationships within the dataset. The user-friendly interface facilitates seamless interaction, enabling users to upload documents, view plagiarism reports, and explore clustered results intuitively. Overall, this project not only addresses the critical issue of plagiarism in academic research but also enhances the research process by enabling users to discover relevant literature more easily.

6.2 FUTURE WORK

In future work, focus on enhancing model scalability by integrating distributed computing frameworks like Apache Spark for handling even larger datasets. Additionally, incorporating advanced GNN architectures such as Graph Attention Networks (GAT) could improve the model's ability to capture complex relationships within articles. Expanding the platform to include unsupervised or semi-supervised learning techniques for better handling of unlabelled data is another potential improvement. Fine-tuning models for domain-specific applications and cross-dataset evaluation would also increase the system's generalization and accuracy. Finally, incorporating real-time updates and continuous learning from new articles would ensure the system remains relevant and accurate over time.

APPENDIX

SAMPLE CODE:

```
from flask import Flask, render_template, request, send_from_directory

import hashlib

import matplotlib

matplotlib.use('Agg')

import numpy as np

import pandas as pd

import os

from sklearn.metrics.pairwise import cosine_similarity

import networkx as nx

import random

def compare_texts(input_text, reference_texts, k=5, t=4):

    input_fingerprints = winnowing(input_text, k, t)

    similarity_scores = {}

    for ref in reference_texts:

        ref_text = ref['text']

        ref_fingerprints = winnowing(ref_text, k, t)

        intersection = input_fingerprints.intersection(ref_fingerprints)

        union = input_fingerprints.union(ref_fingerprints)

        similarity_score = len(intersection) / len(union) if union else 0

        similarity_scores[ref_text] = similarity_score

    return similarity_scores
```

```

tokenizer = AutoTokenizer.from_pretrained("sentence-transformers/all-MiniLM-L6-
v2")

model = AutoModel.from_pretrained("sentence-transformers/all-MiniLM-L6-v2")

def calculate_similarity(doc1_embedding, doc2_embeddings):

    similarity_scores = []

    doc1_embedding = np.array(doc1_embedding).reshape(1, -1)

    for embedding in doc2_embeddings:

        embedding = np.array(embedding).reshape(1, -1)

        if doc1_embedding.shape[1] != embedding.shape[1]:

            embedding = embedding[:, :doc1_embedding.shape[1]]

    similarity = cosine_similarity(doc1_embedding, embedding)[0][0]

    similarity_scores.append(similarity)

    return similarity_scores

def build_similarity_graph(documents, threshold=0.3):

    G = nx.Graph()

    for i, doc in enumerate(documents):

        if isinstance(doc, dict):

            doc_text = doc.get('text', "")

        else:

            doc_text = doc

def plot_graph_density(G, similarity_scores_percentage, document_labels):

    top_10_labels = document_labels[:100]

    top_10_scores = similarity_scores_percentage[:100]

```

```

top_10_labels = document_labels[:10]

top_10_scores = similarity_scores_percentage[:10]

density = nx.density(G)

print(f"Graph Density: {density:.4f}")

plt.xticks(rotation=45)

plt.ylim(0, 100)

plt.tight_layout()

plt.savefig(density_plot_path)

plt.close()

return density_plot_path

app.config['UPLOAD_FOLDER'] = 'uploads'

os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

def extract_text_from_file(filepath):

    if filepath.endswith('.txt'):

        with open(filepath, 'r', encoding='utf-8') as file:

            return file.read()

    else:

        return "Unsupported file type"

app.route('/text_analyze', methods=['GET', 'POST'])

    if file.filename == "":

        return "No selected file"

    filepath = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)

    file.save(filepath)

```

```

extracted_text = extract_text_from_file(filepath)

with open('embeddings_with_text1.json', 'r') as json_file:

    embeddings_data = json.load(json_file)

similarity_results = compare_texts(extracted_text, embeddings_data)

if all(score == 0 for score in scores_values):

    maxscore=0

else:

    max_score = max(scores_values)

    print(f"Text with Highest Similarity: {max_text}")

    plot_data = pd.DataFrame({'Reference Text': texts, 'Similarity Score':
scores_values})

    plot_path = os.path.join('static', 'plots', 'similarity_plot.png')

    plt.figure(figsize=(10, 5))

    plt.barh(plot_data['Reference Text'], plot_data['Similarity Score'],
color='skyblue')

    plt.xlabel('Similarity Score')

    plt.title('Similarity Scores with Reference Texts')

    plt.grid(axis='x')

    plt.tight_layout()

    plt.savefig(plot_path)

    plt.close()

similarity_scores=calculate_similarity(doc1_embedding, doc2_embeddings)

print(similarity_scores)

similarity_scores = abs(similarity_scores)

```

```

similarity_scores_percentage = [
    max(0, 10 - score * 10) if (10 - score * 10) >= 1 else 0
    for score in similarity_scores
]

document_labels = [f"Doc {i+1}" for i in range(len(similarity_scores))]

G = nx.Graph()

for i, label in enumerate(document_labels):

    min_similarity_doc = document_labels[min_similarity_index

    graph_path = os.path.join('static', 'plots', 'similarity_graph.png')

    pos = nx.spring_layout(G)

    node_colors = [G.nodes[node]["color"] for node in G.nodes]

    plt.figure(figsize=(12, 8))

    nx.draw(

        font_size=10, font_weight="bold", edge_color="gray"
    )

    for edge in G.edges(data=True):

        if edge[2]['weight'] == min_similarity_score:

            nx.draw_networkx_edges(G, pos, edgelist=[edge], edge_color='red',
width=2)

            nx.draw_networkx_edge_labels(

                G, pos, edge_labels={(i, j): f"{w:.2f}%" for (i, j), w in edge_labels.items()})

            )

    plt.title("Similarity Graph")

```



```

plt.axis("off")

plt.tight_layout()

plt.savefig(graph_path)

plt.close()

top_10_labels = [extracted_text] + [f"Doc {i+1}" for i in range(10)]

plt.figure(figsize=(12, 6))

plt.bar(top_10_labels, top_10_scores, color=["red"] + ["skyblue"]
len(similarity_scores_percentage[:10])) # Red for your paragraph, blue for others

plt.axhline(y=50, color='red', linestyle='--', label='50% Similarity Threshold')

plt.xlabel('Document Paragraphs')

plt.title('Similarity Percentage Difference (Your Paragraph vs Top 10 Embedding
Paragraphs)')

plt.legend()

plt.tight_layout()

similarity_plot_path=os.path.join('static','plots', 'similarity_percentage_plot.png')

plt.savefig(similarity_plot_path)

plt.close()

return render_template('dashboard.html',

plot_image='plots/similarity_plot.png',

graph_image='plots/similarity_graph.png',

density_image='plots/graph_density_plot.png', maxscore=maxscore)

return render_template('index.html')

def send_image(filename):

return send_from_directory(os.path.join('static', 'plots'), filename)

```

```

database = "database.db"

cur.execute("CREATE TABLE IF NOT EXISTS register (id integer primary key
autoincrement, name TEXT, email TEXT, password TEXT)")

conn.commit()

    return render_template('register.html')

app.route('/dashboard')

def dashboard():

    return render_template('dashboard.html')

app.route('/register', methods=['GET', 'POST'])

def register():

    if request.method == 'POST':

        password = request.form['password']

        conn = sqlite3.connect(database)

        cur = conn.cursor()

        cur.execute("select * from register where email = ? ",(email,))

        existing_user = cur.fetchone()

        return render_template('register.html', message=message)

        cur.execute("INSERT INTO register (name, email, password) VALUES (?, ?, ?)",
(name, email, password))

        conn.commit()

        return render_template('register.html')

app.route('/loginpage')

def loginpage():

    return render_template('login.html')

```

```

app.route('/registerpage')

def registerpage():

    return render_template('register.html')

app.route('/login', methods=['GET', 'POST'])

def login():

    if request.method == 'POST':

        email = request.form['email']

        password = request.form['password']

        conn = sqlite3.connect(database)

        cur = conn.cursor()

        data = cur.fetchone()

        if data:

            return render_template("dashboard.html")

        else:

            message= 'Incorrect Email or Password'

if __name__ == '__main__':

    app.run(port=800, threaded=False)

```

REFERENCES

- [1]. M. F. Manzoor, M. S. Farooq, M. Haseeb, U. Farooq, S. Khalid and A. Abid, "Exploring the Landscape of Intrinsic Plagiarism Detection: Benchmarks, Techniques, Evolution, and Challenges," in IEEE Access, vol. 11, pp. 140519-140545, 2023, doi:10.1109/ACCESS.2023.3338855.
- [2]. H. Cheers, Y. Lin and S. P. Smith, "Academic Source Code Plagiarism Detection by Measuring Program Behavioral Similarity," in IEEE Access, vol. 9, pp. 50391-50412, 2021, doi:10.1109/ACCESS.2021.3069367.
- [3]. Y. Liang, Y. Zhang, D. Gao and Q. Xu, "An End-to-End Multiplex Graph Neural Network for Graph Representation Learning," in IEEE Access, vol. 9, pp. 58861-58869, 2021, doi:10.1109/ACCESS.2021.3070690.
- [4]. W. Yao, K. Guo, Y. Hou and X. Li, "Hierarchical Structure-Feature Aware Graph Neural Network for Node Classification," in IEEE Access, vol. 10, pp. 36846-36855, 2022, doi:10.1109/ACCESS.2022.3164691.
- [5]. R. Jovanovic, M. Palk, S. Bayhan and S. Voss, "Applying Graph Neural Networks to the Decision Version of Graph Combinatorial Optimization Problems," in IEEE Access, vol. 11, pp. 38534-38547, 2023, doi: 10.1109/ACCESS.2023.3268212.
- [6]. L. Almuqren and A. Cristea, "AraCust: A Saudi telecom tweets corpus for sentiment analysis," PeerJ Comput. Sci., vol. 7, p. e510, May 2021, doi: 10.7717/peerj-cs.510.
- [7]. N. Lilian and J. Chukwuere, "The attitude of students towards plagiarism in online learning: A narrative literature review education view project meet African scholars (MAS) view project," Res. Publications, vol. 18, pp. 14675–14688, Aug. 2020.

- [8]. T. Szandała, “Review and comparison of commonly used activation functions for deep neural networks,” in *Bio-inspired Neuro computing*. Berlin, Germany: Springer, 2021, pp. 203–224.
- [9]. S. Boettcher, “Inability of a graph neural network heuristic to outperform greedy algorithms in solving combinatorial optimization problems,” *Nature Mach. Intell.*, vol. 5, pp. 24–25, Jan.2022.
- [10]. K. Langedal, J. Langguth, F. Manne, and D. T. Schroeder, “Efficient minimum weight vertex covers heuristics using graph neural networks,” in *Proc. 20th Int. Symp. Experim. Algorithms*, 2022, pp. 1–17.
- [11]. S. Wang, J. Li, X. Yu, and Z. Xu, "A Comprehensive Survey on Graph Neural Networks: From Basics to Applications," in *IEEE Access*, vol. 11, pp. 101217-101236, 2023, doi: 10.1109/ACCESS.2023.3265499.
- [12]. J. Zhang, Z. Li, and M. Zhang, "Graph Neural Network-Based Drug Repurposing with Graph Embeddings," in *IEEE Access*, vol. 9, pp. 13687-13701, 2021, doi: 10.1109/ACCESS.2021.3054720.
- [13]. L. Chen, Z. Liu, and X. Wang, "Graph Attention Networks for Recommendation Systems," in *IEEE Access*, vol. 10, pp. 14102-14110, 2022, doi: 10.1109/ACCESS.2022.3179267.
- [14]. T. Li, Y. Ma, W. Li, and Y. Zhang, "Hierarchical Graph Neural Networks for Document Classification," in *IEEE Access*, vol. 10, pp. 10223-10231, 2022, doi: 10.1109/ACCESS.2022.3162842.
- [15]. P. Zhang, X. Liu, and L. Gao, "A Survey of Graph Neural Networks in Knowledge Graphs and Their Applications," in *IEEE Access*, vol. 10, pp. 29856-29868, 2022, doi: 10.1109/ACCESS.2022.3165781.
- [16]. M. R. Gupta, V. B. Doshi, and M. E. Gumbau, "Plagiarism Detection in Large-Scale Programming Contests Using Graph Neural Networks," in *IEEE Transactions on*

Knowledge and Data Engineering, vol. 35, no. 7, pp. 1362-1375, 2023, doi: 10.1109/TKDE.2023.3237081.

[17]. X. Zhang, H. Wang, and Z. Zhang, "Improving Graph Neural Networks for Node Classification via Graph Augmentation Techniques," in IEEE Access, vol. 8, pp. 102512-102522, 2020, doi: 10.1109/ACCESS.2020.3015911.

[18]. K. D. Chang, "Survey of Graph Neural Network-based Models for Time Series Forecasting," in Computational Intelligence and Neuroscience, vol. 2022, Article ID 8922061, 2022, doi: 10.1155/2022/8922061.

[19]. Y. Zheng, Z. Xu, and J. Liu, "Optimization Problems in Graph Neural Networks: Algorithms and Applications," in IEEE Transactions on Neural Networks and Learning Systems, vol. 34, no. 3, pp. 705-718, 2023, doi: 10.1109/TNNLS.2022.3180919.

[20]. R. Kumar, A. S. Raj, and S. Kumar, "Graph Neural Networks for Real-Time Anomaly Detection in Streaming Data," in Proceedings of the 2021 IEEE International Conference on Data Mining (ICDM), pp. 937-946, 2021, doi: 10.1109/ICDM51629.2021.00109.

LIST OF PUBLICATIONS AND CONFERENCES

1. D Kavitha, Krishna Prasad H, Manisha R, Rahul R presented on the conference, “International E-Conference on Computational Intelligence and Communication Networks (E-ICCICN’22)” held on 16.3.25.



