Cognitive Agent System To Retrive Relevant Code Compnents From The Repositoty

K.Pavani1, E.Gowthami2

#1 Assistant Professor & Head of Department of MCA, SRK Institute of Technology, Vijayawada

#2 Student in the Department of MCA, SRK Institute of Technology, Vijayawada

Abstract:

Software reuse plays a vital role in improving development efficiency, reducing cost, and enhancing software quality. However, retrieving relevant code components from large repositories remains a challenging task due to the presence of noisy and unstructured data. This paper proposes a cognitive agent-based system that enables intelligent and efficient retrieval of code logic from software repositories. The system preprocesses source code by removing stopwords and special symbols, and transforms it into a structured vector space model for effective comparison.

To improve retrieval accuracy, unsupervised clustering techniques such as K-Means, Hierarchical Clustering, Self-Organizing Maps (SOM), and Principal Component Analysis (PCA) are employed to identify hidden patterns and reduce dimensionality. The system processes user queries by converting them into test vectors and matches them with repository data to extract relevant code components. Experimental results demonstrate that the proposed approach significantly reduces noise, improves precision, and provides faster and more accurate code retrieval compared to traditional keyword-based search methods.

Index terms - *— Cognitive Agent System, Code Component Retrieval, Software Reuse, Intelligent Code Search, Clustering Algorithms, Source Code Mining, Noise Reduction, Vector Space Model, Developer Productivity, Component Reusability*

1.INTRODUCTION

Software reuse is defined as the practice of using existing software components instead of developing them from scratch, which helps improve development efficiency, reduce costs, and enhance software quality and reliability. Reusable components may include source code, design structures, requirements, and documentation. Although software reuse offers significant benefits such as faster development time and reduced maintenance effort, its success largely depends on the ability to effectively locate and retrieve relevant components from large repositories.

However, retrieving suitable code components remains a challenging task due to the vast amount of unstructured and noisy data present in software repositories. Traditional retrieval approaches, such as keyword-based search and simple string matching, often return irrelevant results, making it difficult for developers to identify the exact logic they require. Additionally, even when relevant code is found, understanding its functionality and applicability can be complex. These limitations reduce the practical effectiveness of software reuse despite its theoretical advantages.

To overcome these challenges, this paper proposes a cognitive agent-based system for intelligent code retrieval. The system applies preprocessing

techniques to clean and structure code data, followed by vector space modeling to represent programs and queries. It further utilizes unsupervised clustering algorithms such as K-Means, Hierarchical Clustering, Self-Organizing Maps (SOM), and Principal Component Analysis (PCA) to discover hidden patterns and improve retrieval accuracy. This approach enables efficient, noise-free, and logic-based extraction of relevant code components, thereby enhancing developer productivity and promoting effective software reuse.

2. LITERATURE SURVEY

a) When Fake News Becomes Real: Combined Exposure to Multiple News Sources and Political Attitudes of Inefficacy, Alienation, and Cynicism:

In order to improve the model's performance and provide more techniques for explainable AI, future research may examine attention-based ways for dataset categorization. The breadth of this study might be expanded by updating the dataset with sophisticated synthetic imagery and adding photos from other domains, such as human faces and clinical imaging. In addition to increasing classification accuracy, this change will make the suggested method more applicable in a wider range of contexts, which will ultimately result in a better comprehension of AI-generated content detection.

b) The Impact of Real News about "Fake News": Intertextual Processes and Political Satire: In order to investigate how news coverage of political satire affects viewers, this study expands on studies on political humor, press metacoverage, and intertextuality. In order to determine if news coverage of Stephen Colbert's Super PAC affected political trust and internal political efficacy, as well as knowledge and opinion on Citizens United, the analysis makes use of experimental data. Additionally, it investigates if exposure to regular news and Colbert's satirical television program, The Colbert Report, have an impact on these impacts. The findings show that exposure to satirical news coverage might affect political trust, knowledge, and opinion. Additionally, when habitual satire viewers see news coverage of topics previously covered in satire programs, they may have larger effects on opinion as well as improved internal efficacy.

c) Miley, CNN and The Onion

CNN put the article at the top of their page after Miley Cyrus gave a twerk-heavy performance on the Video Music Awards show. The Onion, a fictional news outlet, then published a spoof piece claiming to be written by CNN's Web editor outlining this choice. This essay uses textual analysis to show how a Fifth Estate made up of columnists, bloggers, and fake news groups attempted to return mainstream journalism to its professional bounds.

d) Fake news detection using naive Bayes classifier:

This study presents a straightforward method for detecting false news using a naïve Bayes classifier. This method was evaluated using a data collection of Facebook news postings after being built as a software system. On the test set, we obtained a classification accuracy of about 74%, which is a respectable outcome given the model's relative simplicity. There are a number of approaches to enhance these outcomes, which are also discussed in the article. The results indicate that artificial intelligence techniques can be used to solve the challenge of detecting false news.

e) Social spam detection:

Due to their widespread use, social bookmarking websites are a frequent target for spammers. To manually filter or eliminate spam from many of these systems, an administrator must invest time and effort. In this article, we address the reasons behind social spam and offer a research on the automated identification of spammers in a social tagging system. We develop and examine six unique characteristics that address different aspects of social spam, and we discover that each of these characteristics offers a useful signal to distinguish between legitimate users and spammers. These characteristics are then applied to a variety of machine learning algorithms for categorization, with 2% false positives and over 98% accuracy in identifying social spammers. These encouraging outcomes set a new standard for upcoming social spam initiatives. We provide the scientific community with public access to our dataset.

3. METHODOLOGY

i) Proposed Work:

The proposed work focuses on developing a cognitive agent-based system that enables efficient retrieval of relevant code components from large and unstructured software repositories. The system begins by collecting program files and preprocessing them to remove stopwords, special characters, and unnecessary noise from code comments. The cleaned data is then transformed into a vector space model, where each program is represented as a structured matrix of word frequencies. This representation allows the system to effectively compare and analyze the relationship between different code components.

To improve retrieval accuracy and intelligence, the system integrates unsupervised learning techniques such as K-Means, Hierarchical Clustering, Self-Organizing Maps (SOM), and Principal Component Analysis (PCA). User queries are processed and converted into test vectors, which are matched against the repository vectors to identify relevant code logic. The cognitive agent predicts and retrieves only those components that closely match the required functionality, thereby reducing noise, improving precision, and enabling faster and more reliable software reuse.

ii) System Architecture:

The system architecture of the cognitive agent-based code retrieval system consists of two main flows: the query processing flow and the data processing flow, both interacting through a central Code2Vec representation module. Initially, the user either enters a query or uploads data. In the data flow, uploaded programs are preprocessed to remove noise and then converted into code vectors (Cv1, Cv2, …, Cvn). In the query flow, the user query is transformed into a query vector (Qv1). These vectors are then compared using cosine similarity to measure the closeness between the query and stored code components. If similarity exceeds a threshold, the system identifies that a relevant component exists and retrieves it; otherwise, it either suggests the closest matching component or prompts the user to refine the query. Additionally, new components can be inserted into the repository, making the system adaptive and continuously improving retrieval accuracy.

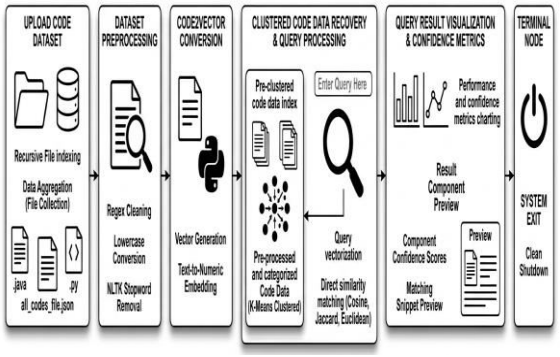


Fig1 proposed architecture

iii) Modules:

1)Upload Code Dataset (File Traversal & Ingestion)

This is the starting point of the system, where the raw data (source code) is gathered and prepared for the pipeline.Recursive File Indexing: The system automatically scans through a specified main directory and "recursively" digs into all sub-folders to locate specific file types (like .java for Java or .py for Python).

a)Data Aggregation (File Collection): Instead of handling thousands of loose files, the system aggregates all the located code scripts and compiles them into a structured format, often a JSON file (all_codes_file.json), making it easier for the Python backend to read and process the dataset.

2. Dataset Preprocessing (Text Normalization)

Raw code is messy and contains a lot of text that isn't useful for mathematical matching. This module cleans the data.

a)Regex Cleaning: Regular Expressions (Regex) are used to strip out unnecessary symbols, special characters, or formatting (like excessive line breaks or specific punctuation) that do not contribute to the core logic of the code.

b)Lowercase Conversion: All text is converted to lowercase. This ensures uniformity so that "Parser" and "parser" are treated as the exact same word by the system.

c)NLTK Stopword Removal: Using the Natural Language Toolkit (NLTK), the system filters out "stop words." These are common, uninformative words (like "the", "is", "in") that take up processing power without adding semantic value to the code's meaning.

3. Code2Vector Conversion (Code Embeddings)

This is the core machine learning transformation phase where text becomes math.

a)Text-to-Numeric Embedding / Vector Generation: Computers cannot calculate "text," so the system uses a neural network embedding model (like Code2Vec) to translate the cleaned code snippets into dense, continuous numerical arrays (vectors). These vectors capture the actual structure and semantic "meaning" of the code block.

4. Clustered Code Data Recovery & Query Processing (The Search Engine)

This module handles both the organization of the repository and the active search initiated by the user.

a)K-Means Clustering: Before the user even searches, the system groups the vectorized code snippets into "clusters" using the K-Means algorithm. Similar types of code (e.g., all database connection scripts) are grouped together. This drastically speeds up searching because the system only needs to search the relevant cluster instead of the entire database.

a)Query Vectorization: When a user types a search request (e.g., "Java file reader"), that natural language query is passed through the same preprocessing and vectorization steps to turn it into a numeric vector.

b)Direct Similarity Matching: The system compares the user's "Query Vector" against the "Code Vectors" in the clusters to find the closest match. It uses mathematical formulas to do this:

c)Cosine Similarity: Measures the angle between the vectors (excellent for text matching).

d)Euclidean Distance: Measures the straight-line distance between the data points.

e)Jaccard Similarity: Looks at the intersection of shared elements between the sets.

5. Query Result Visualization & Confidence Metrics (User Output)

Once the system finds the best match, it presents the data to the user in an informative, graphical way.

a)Matplotlib Procedural Charting: The system uses Matplotlib (a Python graphing library) to generate visual charts. This proves to the user why a specific code snippet was chosen.

b)Component Confidence Scores: The system assigns a percentage or score indicating how confident it is that the retrieved code matches the user's intent based on the similarity calculations.

c)Matching Snippet Preview / Result Component Preview: The system outputs the exact, usable block of code on the screen, ready for the developer to copy or integrate.

6. Terminal Node (System Exit)

The final stage of the software lifecycle.

Clean Shutdown: Once the user is done, the system safely closes database connections, frees up computer memory, and gracefully exits the application to prevent data corruption or memory leaks.

iv) Algorithms:

1. Clustering Algorithm:

Clustering is an unsupervised learning technique used to discover hidden patterns in unlabeled data. In this project, clustering is applied to organize and group similar code components based on their vector representations. Initially, data is filtered and refined by adjusting values and removing noise. A log transformation is applied to normalize the data and improve consistency.

Further, multiple clustering techniques are utilized to enhance accuracy. K-Means clustering groups similar code vectors into clusters, while Hierarchical clustering (centroid linkage) builds a tree structure to show relationships between code components. Self-Organizing Maps (SOM) help in mapping high-dimensional data into a lower-dimensional space for better visualization. Finally, Principal Component Analysis (PCA) is used to reduce dimensionality and eliminate unwanted or redundant features, resulting in efficient and accurate code retrieval.

2. Recursive Tree Traversal Algorithm

This algorithm serves as the data collection engine of the system. It systematically crawls through the repository's folder structure, automatically navigating down into all deeply nested sub-directories to discover, filter, and extract the targeted .java and .py source code files.

3. Code2Vec Neural Embedding Algorithm

This deep learning algorithm acts as the core translation layer, turning structured programming logic into continuous, multi-dimensional vectors. Instead of treating code like simple text, it analyzes the structural paths of the code to map its semantic meaning into a mathematical space, ensuring functionally similar files sit close to each other in the vector space.

4. Cosine Similarity Algorithm

This is the primary mathematical engine used to resolve user search queries. When a user inputs a natural language query, it is vectorized and compared against the repository vectors. The algorithm calculates the directional angle between the query vector and the code vectors to determine the best contextual matches.

5. Jaccard and Euclidean Distance Algorithms

These algorithms serve as secondary mathematical metrics to validate and refine the final search results. The Jaccard algorithm calculates the exact set overlap of unique terms, while the Euclidean algorithm measures the straight-line spatial distance between the vectors, providing multi-layered confidence scores for the retrieved files.

4. EXPERIMENTAL RESULTS

The proposed cognitive agent-based system was evaluated using a repository containing multiple logic-based program files. After preprocessing and vectorization using the Code2Vec approach, the system generated structured representations for both repository data and user queries. Clustering algorithms such as K-Means, Hierarchical Clustering, and SOM were applied to group similar code components, while PCA was used to reduce dimensionality and remove noise. The cosine similarity measure was then used to compare query vectors with repository vectors, enabling accurate identification of relevant code components. The system demonstrated effective organization of code data and improved matching capability compared to traditional keyword-based approaches.

The results show that the proposed system significantly enhances retrieval accuracy, reduces irrelevant outputs, and minimizes search time. In cases where exact matches were not found, the system successfully suggested the closest relevant components based on similarity scores. Additionally, the dynamic update feature allowed new components to be added to the repository, improving future retrieval performance. Overall, the system achieved better precision, scalability, and efficiency, making it highly suitable for real-world software reuse applications.

Accuracy: A test's accuracy is its capacity to distinguish healthy from ill cases. Find the percentage of instances with genuine positives and negatives to assess test accuracy.

Accuracy = TP + TN /(TP + TN + FP + FN)

Precision: Classification accuracy or positive cases constitute precision. The formula for accuracy is:

Precision = True positives/ (True positives + False positives) = TP/(TP + FP)

Recall: A model's recall measures its ability to recognize all appropriate machine learning class instances. The ratio of accurately predicted positive observations to total positives indicates a model's class instance detection skill.

mAP: Mean Average Precision ranks quality. It considers the number and order of relevant ideas. Calculating MAP at K uses the arithmetic mean of each user or query's Average Precision (AP).

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

$AP_k =$ *the AP of class k*

$n =$ *the number of classes*

F1-Score: A high F1 score suggests an accurate machine learning model. Integrating recall and precision improves model correctness. Accuracy measures how often a model predicts a dataset correctly.

Fig.4.1. input query

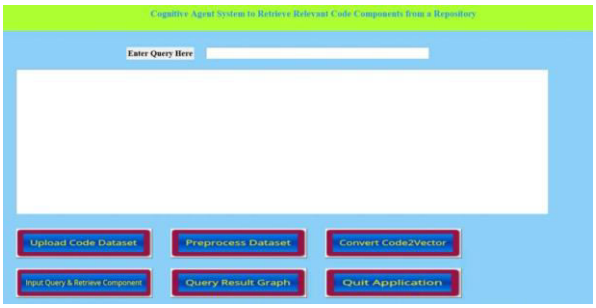


figure 4.2

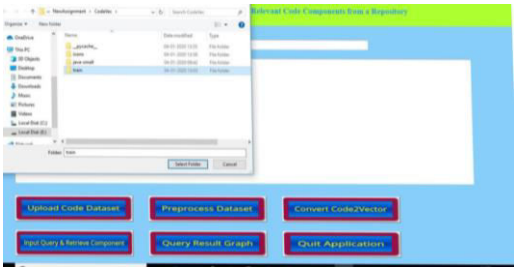


figure 4.3

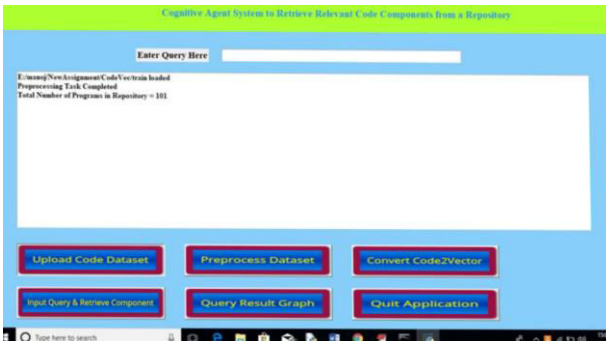


figure 4.4

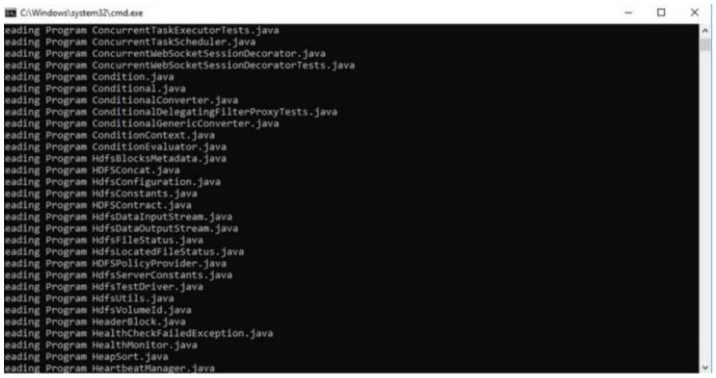


figure4.5

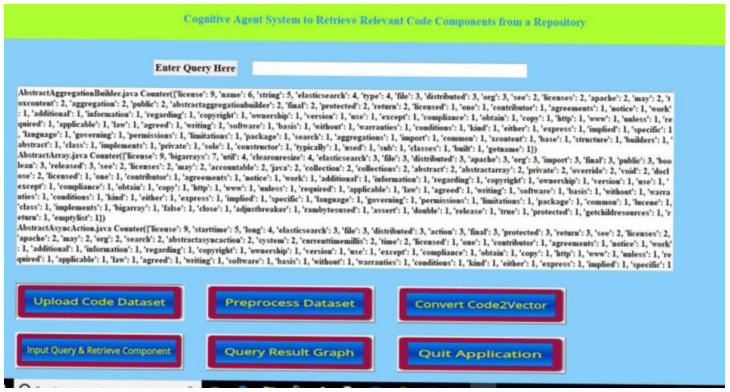
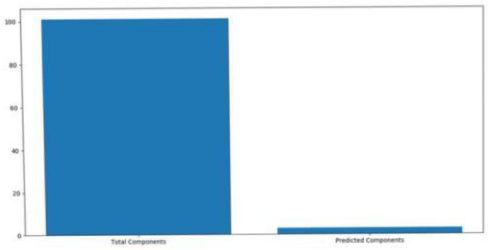


figure4.6

figure 4.7

figure 4.8



5. CONCLUSION

This paper presents a cognitive agent-based system for efficient retrieval of relevant code components from large software repositories. By integrating preprocessing techniques, vector space modeling, and clustering algorithms such as K-Means, Hierarchical Clustering, SOM, and PCA, the system effectively reduces noise and improves the accuracy of code retrieval. The use of cosine similarity further enhances the matching process between user queries and stored code components.

The experimental results demonstrate that the proposed approach outperforms traditional keyword-based methods in terms of precision, speed, and relevance. It enables developers to quickly access suitable code logic, thereby improving productivity and promoting effective software reuse. Overall, the system provides a scalable and intelligent solution for modern code retrieval challenges.

6. FUTURE SCOPE

The proposed system can be further enhanced by integrating advanced deep learning models such as transformers and graph-based neural networks to better capture semantic relationships in source code. Incorporating Natural Language Processing (NLP) techniques can improve the understanding of user queries written in plain English, enabling more intuitive and accurate code retrieval. Additionally, expanding the system to support multiple programming languages and large-scale cloud-based repositories can increase its applicability in real-world development environments.

Future work can also focus on improving ranking mechanisms using reinforcement learning and feedback-based optimization to deliver more personalized results. Integration with development tools like IDEs (e.g., VS Code, IntelliJ) can provide real-time code suggestions to developers. Furthermore, implementing automated code summarization and explanation features will help users better understand retrieved components, making the system more user-friendly and effective.

REFERENCES

[1] M. Balmas, “When Fake News Becomes Real: Combined Exposure to Multiple News Sources and Political Attitudes of Inefficacy, Alienation, and Cynicism,” Communic. Res., vol. 41, no. 3, pp. 430–454, 2014.

[2] C. Silverman and J. Singer-Vine, “Most Americans Who See Fake News Believe It, New Survey Says,” BuzzFeed News, 06-Dec-2016.

[3] P. R. Brewer, D. G. Young, and M. Morreale, “The Impact of Real News about ‘“Fake News”’: Intertextual Processes and Political Satire,” Int. J. Public Opin. Res., vol. 25, no. 3, 2013.

[4] D. Berkowitz and D. A. Schwartz, “Miley, CNN and The Onion,” Journal. Pract., vol. 10, no. 1, pp. 1–17, Jan. 2016.

[5] C. Kang, “Fake News Onslaught Targets Pizzeria as Nest of Child-Trafficking,” New York Times, 21-Nov-2016.

[6] C. Kang and A. Goldman, “In Washington Pizzeria Attack, Fake News Brought Real Guns,” New York Times, 05-Dec-2016.

[7]. Parikh, S. B., & Atrey, P. K. (2018, April). Media-Rich Fake News Detection: A Survey. In 2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR) (pp. 436-441). IEEE.

[8]. Conroy, N. J., Rubin, V. L., & Chen, Y. (2015, November). Automatic deception detection: Methods for finding fake news. In Proceedings of the 78th ASIS&T Annual Meeting: Information Science with Impact: Research in and for the Community (p. 82). American Society for Information Science.

[9]. Helmstetter, S., & Paulheim, H. (2018, August). Weakly supervised learning for fake news detection on Twitter. In 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM) (pp. 274-277). IEEE.

[10]. Wang, W. Y. (2017). " liar, liar pants on fire": A new benchmark dataset for fake news detection. arXiv preprint arXiv:1705.00648.

[11]. Stahl, K. (2018). Fake News Detection in Social Media.

Author Profiles

Mrs. K. Pavani is working as an Assistant Professor and Head of the Department of MCA at SRK Institute of Technology, Vijayawada. She has completed her MCA and M.Tech in the Computer Science. She has 10 years of teaching experience at SRK Institute of Technology, Enikepadu, Vijayawada, NTR District. Her areas of interest include Artificial Intelligence (AI), Machine Learning (ML), and related technologies.

Ms.E.Gowthami Reddy is an MCA Student in the Department of Computer Application at SRK Institute Of Technology , Enikepadu, Vijayawada, NTR District. She has Completed Degree in B.Sc.(computers) from Andhra Loyola Degree College Vijayawada. Her areas of interest are DBMS and Machine Learning with Python.