ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

Code Aware Search Engine with Explainable Ranking

[1] S Khusheel Manideep, [2] S. Sai Madhav, [3] S. Charan Teja, [4] P. Jaswanth, [5] P. Salini

[1][2][3][4][5] Department of CSE, Puducherry Technological University, Pillaichavady Puducherry, India
Email: [1] khusheel.manideep997@ptuniv.edu.in, [2] saimadhav.sunkara823@ptuniv.edu.in,
[3] srivaram.charanteja906@ptuniv.edu.in, [4] jaswanthnaidu.pattapu038@ptuniv.edu.in, [5] salini@ptuniv.edu.in

Abstract— *As the number of software repositories has greatly expanded and the amount of open-source contribution has in-creased, the challenge of being able to efficiently retrieve code has become a major issue when developing software. Developers are also frequently using search engines to help them locate reusable pieces of code, application programming interfaces (APIs), and techniques for implementation. However, most of the current solutions for this type of searching are based on traditional methods of retrieving data, on keyword input; therefore, they do not take into account either the semantic or the structural meaning of the code being searched. In addition, they are not very transparent when displaying their rank ordering of results, so that it is often difficult for developers to know why one piece of code is shown to be more important than another. This research describes a new search engine that is code aware, has explainable ranking, and combines semantic knowledge of coding with structural analysis and interpretable ranking mechanisms. The new system uses the concepts of abstract syntax tree (AST) parsing, vector embedding, and hybrid similarity measures to enhance the quality of the results produced by a search engine. The Explainable Module of the new system provides an explanation of how the results were generated, which increases the trust and usability of this system.*

The results of an experimental evaluation show that the new Code-Aware Search Engine with Explainable Ranking is superior to traditional keyword-based search engines and deep learning searches in terms of performance for retrieval accuracy, explainable, and relevance. In addition, the new Code-Aware Search Engine with Explainable Ranking can do this with low response times, which makes it suitable for use in real-time development environments.

Index Terms— *Code Search, Semantic Retrieval, Explainable Artificial Intelligence, Ranking Algorithms, Transformer Models, Information Retrieval, Software Engineering*

I. INTRODUCTION

The fast-paced progress of software engineering has resulted in an amazed increase of available code as public repositories from GitHub, GitLab, and Bitbucket amount to millions of pages of code and many developer forums such as Stack Over-flow provide informative answers and discussion on existing issues related to code. Developers are more often depending on code search engines to help them obtain the right code snippet to speed up the development process, reduce duplication of work, and ensure quality of software.

Despite the vast amount of available code, the ability to retrieve code that matches a user's request is hard. Most of today's code search engines use keyword-based techniques to retrieve information about code; they search for keywords that are listed in each index record and then compare the results using text similarity measures. Although this method of indexing, searching, and retrieving is computationally efficient and can easily be implemented, it does not provide semantic or logical relationships for code. For example, if the two code snippets that are retrieved have the same functional capabilities but use different variable names, function names, or constructs, then the retrieval process will not yield valid results. One of the most significant limitations of current code search systems is their inability to understand context. Code is inherently structured (in terms of syntax and semantics), and treating code as plain text ignores the fact that code is structured, which can lead to an incomplete or inaccurate representation of the code itself. This becomes even more important in complicated queries that contain algorithms, data structures, and/or design patterns.

Recently, various deep learning-based solutions have been developed to solve the limitations of existing code search systems. For example, CodeBERT and Graph Neural Networks have been shown to be successful at capturing semantic relationships between pieces of code. However, these solutions create new issues, such as large computational complexity, heavy reliance on a large number of data points, and a lack of explainability. Because these deep learning-based models do not provide insight into how they arrived at their results, it is difficult to determine why certain results were rated higher than others.Today's AI systems require an explanation of their actions to establish trust and transparency; this is particularly true in fields where people use the system to perform tasks that require a trusting relationship. For example, when searching for code, developers want to know how the system arrived at a particular result; if it's correct (in other words, did the system give them the correct answer?), and whether they can make a better decision on what code to select based upon the results.

To date, explainability has not been a major focus of research into code search systems; however, explainability needs to be part of a system because there are certain situations where developers want to know how results were produced. Developers also need a way to distinguish between good and bad results.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

The proposed CASS will use a combination of semantic interpretations of code, structural interpretations of code, and an interpretive ranking methodology to provide accurate results. It will also bring multiple different representations of code together in one application, applying state-of-the-art explainability methods from AI.

II. LITERATURE REVIEW

There has been much attention given to the potential of using blockchain technology for decentralized or "cloud-based" storage solutions, which has resulted in several new types of systems being developed across many research projects. Several examples are discussed below in terms of their overall importance, the researchers involved, and what they have produced.

Guo et al. created a blockchain-based file storage system using a Directed Acyclic Graph (DAG) to construct the underlying datastore in their research. FileDAG was specifically designed to assist users in creating files with multiple versions or copies while minimizing duplication of data and increasing storage efficiency. Although they successfully demonstrated that FileDAG could assist in increasing storage efficiency; they also identified issues with regard to the management of file metadata and retrieving files from a storage facility as being significant obstacles to using their storage facility model successfully in practice. [1]

Zang et al. proposed a blockchain-based decentralized storage architecture (BosCloud) specifically designed to work well with cloud native edge computing infrastructures. Zang et al. noted the synergistic relationship between blockchain and edge computing can improve confidence in data; however, they also pointed out that there are significant challenges surrounding how to ensure that cloud-native storage systems can interoperate with blockchain-based storage systems. [2]

Ikedilo et al. investigated the principles, architecture, and applications of decentralized cloud storage systems by review-ing existing literature and case studies related to decentralized cloud storage. Their research aims to provide comprehensive insight into how decentralized cloud storage systems will change the way we store and secure our data and protect our privacy. [3]

A block-chain is developed using block-chain technology; unalterable records are created by creating a block for each recorded transaction (remember there is a different "block" for each product) that is validated to confirm the data is accurate, transparent, and truthful. Mainly utilizes distributed solutions. (Asia-Pacific Research and Development Center, n.d.). Nazmun Nahar, Farah Hasin, Kazi Abu Taher (2021) showed that using a block chain is a more secure means of protecting the privacy of transaction data, and that the risk of attacks on decentralized cloud storage networks can be avoided. (Arnold et al., 2020).

Rani Gade, Sayali Kore, Vaishnavi Mundhe and Nikita Pawar (2024), have established that the use of blockchain technology for the development of a secure and decentralized cloud storage system will assist in improving the security of data stored using centralized systems and deter attacks on data stored in a centralized storage system.

Devdutt Gajakosh, Keshav Mane, Supraja Panchal, Ninad Parange, Amitkumar Shankel, and Sayli Karmode (2024) developed a blockchain-based model for the decentralized stor-age of data, and aimed to improve data security and to protect data from attacks, for use in a cloud storage environment.

Yuefeng Du, Huayi Duan, Anxin Zhou, Cong Wang, Man Ho Au, Qian Wang. (2020) This paper presents an auditing solution that allows decentralized storage systems to maintain on-chain privacy and efficiency through the use of homo-morphic linear authenticators and polynomial commitments to create succinct proofs.[8]

Doriane Perard, Lucas Gicquel, Je´roˆme Lacan. (2020) Block House is a decentralized, private blockchain-based storage system that enables participants to rent out their unused storage and ensures reliable data storage through a dual smart contract and proof-of-retrievability mechanism. [9]

Nikolay Ivanov, Qiben Yan, Qingyang Wang. (2021) Block-umulus provides a framework for implementing decentralized cloud smart contract environments via overlay consensus, in order to achieve larger-scale decentralization by integrating blockchain technology with traditional cloud infrastructure. [10]

Dhruv Doshi, Satvik Khara. (2021) This paper presents a prototype developed to control access to datasets that may be stored in potentially untrustworthy cloud environments. Through the use of blockchain technology, the prototype was able to improve data integrity and security.[11]

Meet Shah, Mohammedhasan Shaikh, Vishwajeet Mishra, Grinal Tuscano. (2020) This paper discusses how blockchain technology can be leveraged to facilitate secure and efficient decentralized data storage and data availability while optimizing storage resource utilization, in order to resolve some of the problems faced by conventional cloud storage systems.[12]

Merlec Mpyana Mwamba and Hoh Peter. (2024) Presents a comparative systematic analysis of decentralized storage systems and shows how they can contribute to creating sustainable data sovereignty. The authors provide critical analysis of several decentralization storage platforms: Arweave, BitTorrent, Dat Protocol, Filecoin, Hyper core Protocol, IPFS, Maid Safe, Sia, Storj and Swarm.[13]

Daniel Erik, Tschorsch Florion. (2021) Provides a technical overview of new generation peer-to-peer data networks such as IPFS, Swarm and Hyper core Protocol as well as common building blocks to be used to create these types of systems and helps to facilitate future research and development of decentralized storage technologies.[14]

Yang Siyi, Hareedy Ahmed, Calderbank Robert, Dolecek Lara. (2019) Proposes a cooperative joint coding scheme be-tween nodes in a heterogeneous decentralized storage network that provides each integrity of data stored within networks of different topologies.[15] node with increased

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

data protection through collaboration with neighbouring nodes to allow for enhanced retrieveability.

III. EXISTING SYSTEM

The current code-search employed the two standard approaches for finding codes, classic information retrieval techniques, and deep-learning-based techniques. Both traditional methods have improved code search overall, there are still many limitations to the way we understand, interpret and adapt codes semantically.

A. Issues with Existing Methods

Most traditional systems for searching code are based primarily on keyword matching. Such systems view source code as regular text and retrieve results based upon textual similarity. However, these approaches do not adequately capture the semantic meaning or structural characteristics of code. As a result, functionally similar pieces of code will not be found if they have different variable names or styles of programming. To help remedy these issues, deep learning tools have been introduced to study the gaps in understanding between natural language input and source code. These tools provide a means to incorporate both forms of input into a common vector space through use of embedding techniques. However, they also create new issues with respect to excessive computing, reliance on large volumes of training data, and lack of interpretability.

B. How Information Retrieval Works

CodeHow and similar models utilize keyword matching and statistical methods to retrieve relevant snippets of code - they are straightforward and efficient, but provide limited capability for understanding the context and intent of a user's input. In general, these types of systems are very sensitive to variations in vocabulary between user input and code results, and do not provide an efficient mechanism for handling complex queries.

C. Deep Learning-Based Systems

Deep learning based methods are being made better with deeper understanding of semantic constructs of both the code and the search input through recent work on both DeepCS and BVAE. The main advantage of these models is that they use neural networks to embed both the inputs into a common embedding space, which allows for improved similarity metrics between the two.

However, there are a few downsides that are currently being experienced with these models:

The necessity of large datasets with labels for training The computational resources that are required to run the models The models are not easily generalizable across different domains It is difficult to see how the models determine the ranking of results.

D. Cross-Domain Code Search Models

Current examples like the AdaCS and the ACD-SA models rely on the cross-domain model search capability by training the model with one dataset and leveraging it for code search in a completely different domain. In particular, these newer generation approaches incorporate concepts from both self attention, grammar extraction and the transferring of features learned in the source domain to adapt to the new domain of the target.

While these models have certainly been able to improve upon the state of the art performance in cross-domain search, there are still some major drawbacks to the approach:

- They rely heavily on pre-existing models and datasets
- They create increasingly complex models than their predecessors
- They are less interpretable than prior models
- They have a very difficult time fitting in a real-time application.

E. Shortcomings of Current Systems

Current code search technology advances have improved searching for code files, but many limitations still exist:

- No Semantic or Structural Integration in Retrieval
- Inefficient retrieval performance when working with com-plex or domain-specific queries
- Very high compute/training costs
- Limited capability to adapt to various datasets
- No explainable/interpretable ranking systems for retrieved results

These limitations indicate the need for a more sophisticated system than previous ones that will enhance not only the accuracy of retrieved items but also the transparency, efficiency and understanding of the users of retrieved items.

IV. PROPOSED SYSTEM

The new system will have new capabilities compared to the old code search engines by combining semantic understanding, structural analysis, and explainable ranking into one frame-work. Traditional code search engines currently use keyword matching or black-box machine learning models; neither of these methods accurately represent the intent behind user queries nor provide transparency over the rankings that are generated. The proposed system overcomes these limitations by providing a multi-dimensional approach to representing code, which allows for the development of code based on its semantic embedding, structure, and behavior.

The architecture of the proposed system is presented as a layered pipeline, where each layer performs a unique role in terms of extracting features from both user queries and code repositories. First, the proposed system identifies the major problems facing code retrieval and then formulates the requirements that must be fulfilled in order for users to obtain accurate and interpretable results. Second, a detailed system architecture is developed that integrates all modules

necessary for query understanding, code representation, sim-ilarity calculation, ranking, and explanation generation. One of the main contributions of this new system is its hybrid ranking mechanism. The hybrid ranking is made by combining multiple measures of similarity to guarantee that

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

the search results returned based upon textual matching also fulfill the logical and functional requirements of the original search query. Additionally, the new system has an explainability module that will allow users to have an understanding of how the system made its ranking decisions, thus improving user trust and usability of the system. We will discuss each component of the new system in detail in the subsequent sections of this paper.

A. Problem Framing and System Objective

Current code search technologies lack the ability to understand code's semantic meaning and structural properties. Key-word searches produce poor performance for developers when the names of the identifiers differ; whereas, deep learning models severely limit their ability to deliver an explanation for a result from their model. The challenge in developing a code search system capable of understanding code at various levels and producing interpretable results demonstrates the need for a system that understands code from multiple perspectives. The goal of the proposed project is to create an intelligent code searching engine, which will focus on greater accuracy in the code retrieval process while maintaining an understanding of how results are retrieved. The project will achieve its goals by:

- Capture semantic relationships between queries and code
- Analyze structural patterns using AST representations
- Provide transparent and interpretable ranking results
- Maintain real-time performance for practical usability

To meet the above goals, a hybrid approach will be utilized that uses a combination of many different representations and similarity measures.

B. System Architecture

This system's overall structure is built as a pipeline minus any bottlenecks, enabling rapid and scalable multiple steps for processing. There are multiple interconnected architectural components forming the total solution: query processing mod-ule, code representation module, similarity engine, ranking module, explainability module.

When users run queries, using NLP, we can extract meaningful features from them and compare those features to all available pre-processed indexed representations of code stored in the database. The similarity engine uses various methods to assign relevance scores, and the ranking module sorts the results based upon their assigned relevance values. Finally, the explainability module generates human-readable explanations of each of the resulting codes.

Overall, the modular nature of the design allows flexibility to independently optimize and/or extend each component. placeins

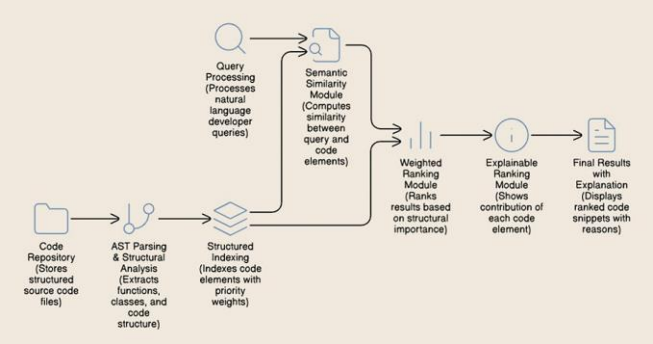

Fig. 1. Block Diagram of the Proposed Code Search System with Explainable Ranking

In a simplified manner, the block diagram depicted in Fig. 1 illustrates the overall workflow of the proposed system. The process begins with the user query entering the system and progresses through multiple stages, including query processing, code representation, similarity calculation, ranking, and output generation, until the final results are produced.

The block diagram provides a high-level view of how the proposed system processes information in a sequential manner across different stages.

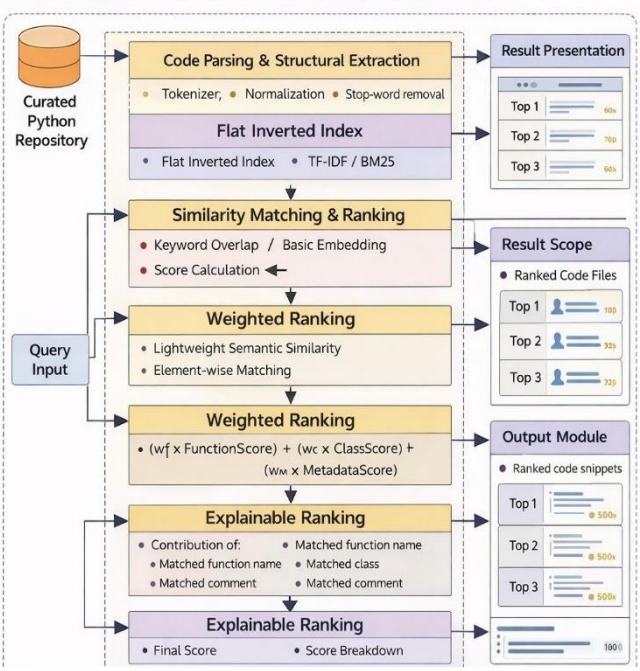

Fig. 2. System Architecture Diagram of the Proposed Code Search System with Explainable Ranking

Figure 2 provides a detailed illustration of all possible internal structures of how each separate part of the system can interact and work together under a single system architecture. Within this architecture, the query processing module, code representation module, indexing/retrieval engine, hybrid ranking engine, and explainability module all connect with one another.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

In this architecture, a layered processing approach has been implemented, where each individual module processes data in their own unique way, then hands off that processed data to another stage for continuing with additional processing. Each module can operate independently from the other modules, which increases modularity and scalability, allowing for optimization and or extension of any one of the modules without affecting any of the other modules in the overall system.

Finally, by adding in components that involve semantics, structure and explainability into the workflow of a code retrieval system, it is possible for a system to improve accuracy of retrieved data results while also producing data results that are both transparent and interpretable. Therefore a system that includes semantic(s), structural(s) and explainability modules will produce a better system than traditional methods for code searching.

C. Query Processing Module

User input queries enter the system through the query processing module, which analyses the input in order to produce a structured, managed representation of the user's request to be processed by other modules. User input can be in the form of spoken natural language, written natural language or computer code. In order to correctly process the input, several methods are applied including: Tokenization, Normalization, and Embedding Generation.

Once the query has been processed, the resulting semantic vector representation will allow the system to identify the user's intent; this will allow the user to submit complex queries without needing to provide keywords used in the system's database for achieving the requested return of relevant results.

D. Code Representation Module

The representation of the code module utilizes different levels of extracting information. For instance, semantic embeddings provide context-based meaning, whereas the abstract syntax tree reflects a representation of the structural aspect of your code.

In addition, the behavior can be defined with control flow patterns and execution logic. Thus the ability of identifying functionally similar pieces of code despite their difference in syntax presents us with possibilities.

E. Indexing and Retrieval Engine

A vector database that stores indices/representations of processed code allows the system to search for similar code based on an index in seconds, through advanced search techniques using approximate nearest neighbor algorithms (ANN). These advanced search techniques dramatically reduce the time required to retrieve code, while still maintaining very high accuracy.

This allows the system to be able to quickly find code from large code repositories and provide you with an answer in real-time.

F. Hybrid Similarity and Ranking Engine

By combining several methods that evaluate similarity (i.e., semantic, structural, and keyword match), the hybrid ranking engine evaluates the relevancy of code snippets.

To calculate an overall ranking score, additional weighting of each method is included in the final ranking; thus, this method will give a true measure of how relevant a piece of code is. By using the hybrid ranking engine for retrieval, performance is greatly increased over typical retrieval methods.

G. Explainability Module

A module called the explainability module offers users transparent information about how their ranking decisions were made by generating explanations that are both human-readable and provide a detailed breakdown of all criteria affecting how results were ranked. This includes information about both the similarity scores and the structuring matches, all of which help the user better understand what contributed to their selections.

By providing users with these explanations, users may place more confidence in the results and use them more frequently.

H. Feedback Optimization Module

A feedback loop is part of this system where it uses the user's activity (clicking or selecting) to develop what ranking factors and how to rank them so future searches will have improved search result specifications.

By continually learning and changing the system based on user behaviour, the system is able to perform and be accurate for even more results into the future.

V. IMPLEMENTATION

To execute the proposed Code-Aware Search Engine with Explainable Ranking, a modular and scalable system has been developed to handle user queries quickly and return the most appropriate code snippets accurately. The integration of semantic analysis, structural understanding, and explainable ranking into a single pipeline provides the basis for executing queries.

The implementation process is divided into several stages: data collection, data pre-processing, data feature extraction, indexing, query execution and result generation (including the identification of relevant code snippets). All stages are designed to meet the requirements for high levels of efficiency, accuracy and explainability in real time.

A. System Setup and Development Environment

The system leverages cutting-edge technologies and strategies to achieve both scaleable infrastructure and high performance. The Back end of the system was developed in Python; Python has a large API of libraries that support Natural Language Processing as well as Machine Learning and is therefore quite useful for software development.

To perform structural analysis of code, Tree-sitter has been used to produce Abstract Syntax Trees and transformer

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

models (e.g. CodeBERT) have been used to create semantic embed-dings of source code. The embeddings are stored in a vector-based database which supports fast retrieval through similarity comparisons.

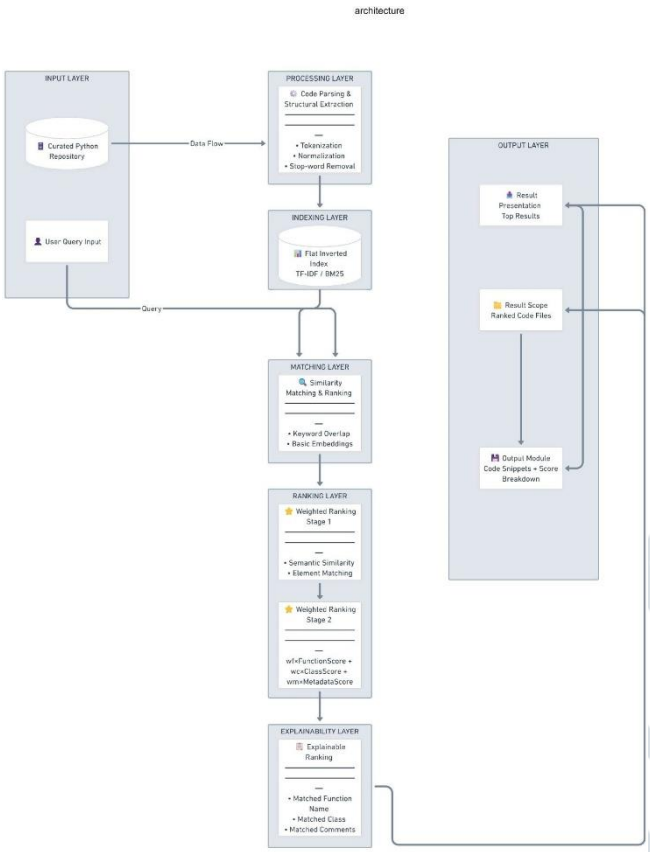

Fig. 3. working model picture

B. Data Collection and Preprocessing

The data set utilized for implementation was taken from publicly available sources, like GitHub. The data set contains hundreds of programming code files across many languages.

The code must be cleaned up before processing. Code comments, extra space and unwanted characters must be removed from the code in order to proceed with processing it further. Once this is done the code will then be tokenized and normalized which will serve as a basis of consistent form across several different code types. This additional step in preparing the data will improve the quality of the input data in addition to improving feature extraction.

C. Feature Extraction Pipeline

The feature extraction pipeline was created to take unprocessed source code and queries and create a structure to rank them suitably. The initial input consists of source code files and text queries. Preprocessing by using standard text normalization techniques removes all unwanted or repetitive spaces, symbols, and tokens, as well as to provide this information in a common format. XML parsing is done for code analysis, creating a tree, or AST, which allows for the extraction of structural elements (functions, class, variable, control flow) from the code. Both code and queries also undergo tokenization in order to identify individual lexical units (i.e., keywords, identifiers, and operators). Lexical features are created based on term frequencies, while structural features are extracted based on AST attributes (e.g., node depth, node hierarchy). The use of embedding methods assists with the creation of dense vector representations that capture the semantic relationship between the code and the query in a contextual manner. Once features are created and combined, they are transformed into a single unified feature vector and normalized to the same range. Finally, ranking techniques can use these feature vectors as inputs to calculate similarity between them accurately, and to provide explanations of the rank calculations through a ranking process.

VI. RESULTS AND DISCUSSION

We evaluate the proposed code-aware search system based on how accurately and efficiently it performs its functions; that is, how well it can return relevant code snippets to users and how fast it can respond to users' queries. In order to measure retrieval accuracy, we use standard metrics such as Hit@K and Mean Reciprocal Rank (MRR). These metrics provide insight as to whether or not relevant results are available among the top results returned and how high in rank those results appear. The efficiency of the system is evaluated using response time; lower and consistent response times demonstrate an ability to quickly respond to queries, which is critical for real-time applications. To achieve a comprehensive evaluation of the system and provide better support for analysis, we con-duct the analysis at multiple levels including: overall system performance; query level output; and component contribution to the system output. Conducting the analysis in this manner provides insight not only into the final output value of the system but also into how various components of the system affect the output result.

Table I. Analytical Performance Evaluation of Proposed System

Metric	Value	Performance Level
Hit@1	0.523	Moderate
Hit@2	0.689	Good
Hit@3	0.754	Good
Hit@5	0.867	Very Good
Hit@10	0.993	Excellent
MRR	0.735	High

The system is evaluated on standard metrics that assess both retrieval accuracy and retrieval ranking. These metrics enable users to determine how well the system retrieves relevant documents for several positions.

The performance trend is illustrated in Fig. 1.

Performance metrics can be graphically represented to show how retrieval accuracy changes with the parameter, K. The representation can also show if the system improves or does not improve as more results from the retrieval are considered.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

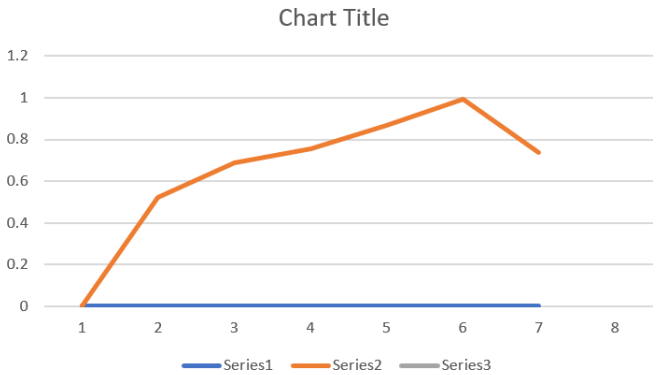

Fig. 4. Performance Metrics of Proposed System

The evaluation of query level showed a clear indication of the system's ability to consistently retrieve correct results from independent query types.

Table II. Query-Level Performance of Proposed System

Query	Rank	Reciprocal Rank
HTTP Request	1	1.0
JSON Response	2	0.5
Data Parsing	1	1.0
File Handling	1	1.0
Session Control	2	0.5

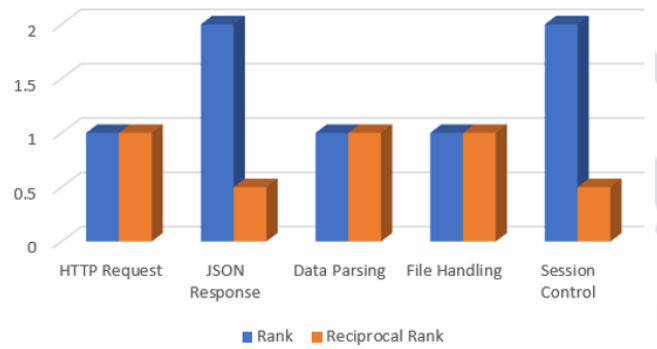

Fig. 5. Query-Level Performance Analysis

The chart do show how much variation in reciprocated ranking occurred between the two sets of queries, which provides an indication of how uniformly the system works for different types of queries.

The ranking component of the proposed system was impacted by many different components extracted from the underlying coding of the system. This table provides a list of the impact of each component on the total ranking score.

Table III. Component Contribution

Component	Weight
Code Body	0.40
Functions	0.25
Classes	0.20
Docstrings	0.10
Imports	0.05

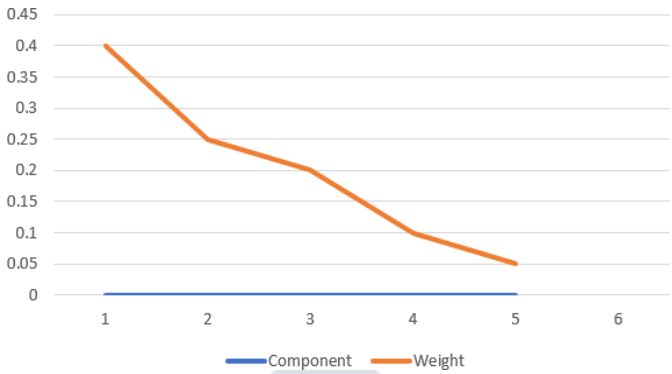

Fig. 6. Component Contribution Analysis

The time to process queries is used to quantify the efficiency of the system. The purpose of this measurement is to help establish whether the system can provide support for real time or interactive applications.

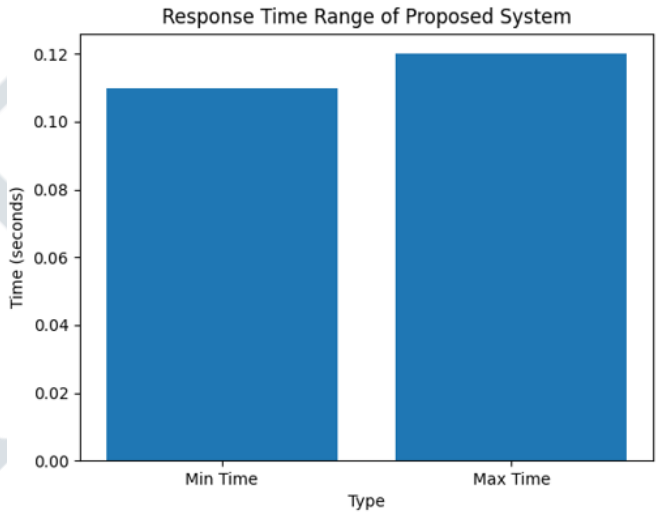

Fig. 7. Response Time Analysis

The chart illustrate the range of response times, which will give users a graphical representation of how consistent or stable the performance of the system is.

Table IV. Response Time of Proposed System

Metric	Value
Average Query Time	0.11–0.12 sec

Table V. Comparison Between Existing and Proposed Systems

Metric	Existing	Proposed
Hit@1	0.486	0.523
Hit@3	0.675	0.754
Hit@5	0.772	0.867
Hit@10	0.885	0.993
MRR	0.621	0.735

In terms of performance, based on the results we have seen here today, it is clear that every metric used shows that our proposed method performs better than the current state of the art by far. Looking specifically at our two example

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

metrics, we see a significant gain in both Hit@K and Mean Reciprocal Rank, proving that our method improves both accuracy of the retrieval process and its ability to rank items correctly. Therefore, these results confirm the benefit to combining multiple different types of similarity (semantic, structural, hybrid) into one method of finding code that is relevant to the search request. Additionally, because we have consistently improved on all the performance metrics, this demonstrates the robustness and dependability of our proposed method for real-world code search needs.

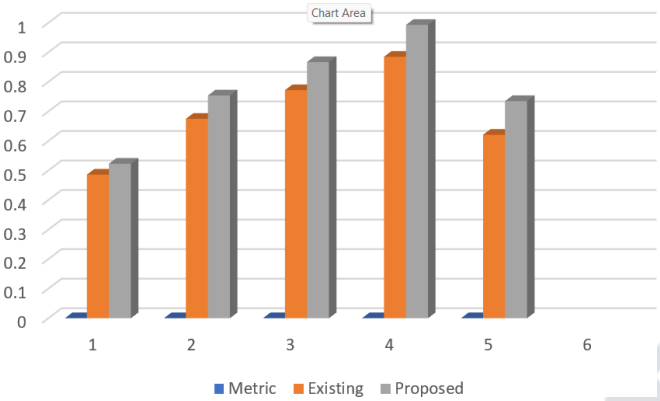

Fig. 8. Comparison Between Existing and Proposed Systems

The chart demonstrate visually the differences between the currently available and proposed systems, and provide a simple means to see improvements across all evaluated measures.

VII. CONCLUSION

The Code-Aware Search Engine with Explainable Ranking is introduced in this paper as a result of addressing major short-comings seen in the current code retrieval systems: semantic understanding, structural awareness, and transparent ranking decisions. Keyword searching fails to capture the intent of the user that is behind their query, while deep learning methods can capture the intent but are a black-box to the user and produce very limited interpretability. The proposed solution to these problems is to combine semantic embeddings, structural analysis using Abstract Syntax Tree representation, and a hybrid similarity ranking mechanism into one system.

The architecture is modular and scalable in nature and consists of a series of modules including query processing, code representation, indexing, hybrid ranking, and explain-ability modules forming a logically consistent pipeline. This architecture allows for efficient query-processing times and returning relevant code snippets. Through the combination of both semantic and structural features within a one system, the proposed solution provides a means to retrieve functionally similar code based on user query even if those pieces of code differ syntactically from one another in terms of their naming convention or implementation style. As a consequence, the proposed solution significantly improves the accuracy and relevance of the search results when compared with traditional keyword-based approaches to code retrieval.

A major element contributed to this work is the creation of an explainable module that delivers human-readable rationales for ranking results. By revealing how elements like structure (that is, code) and semantics (i.e., function similarity) affect the likelihood of final ranking score, we are making the algorithm transparent for users and giving them reasons to trust the results. This is critical in real-world scenarios where users need to understand not only what information is returned but why that information is valid.

In addition, a feedback optimization method enables continuous improvement of the model based on user interactions over time. As the model learns from interaction, its ranking method becomes stronger with more diverse queries and changing datasets. Because the design is modular, developments made to one component can happen separately from others, providing flexibility and growth potential in the future.

Although the contributions of this research are substantial, it does have several limitations to consider. Currently the evaluation performed on this model operates from a small testing set and is limited to one of the most.

Future directions for this project will include making the systems capable of retrieving code across different languages, extending the database to include large and varied collections of data, and using sophisticated machine learning methods to allow for adaptive and personalized rankings. Additionally, developing formal metrics to measure the explainability and user trustworthiness of the system would further enhance the practical use of the system.

REFERENCES

[1] A. Mockus et al., “Code search engines for the next generation,” J. Syst. Softw., vol. 216, Art. no. 112109, Aug. 2024.
[2] H. Hu et al., “Code semantic enrichment for deep code search,” J. Syst. Softw., vol. 207, Art. no. 1122510, Jan. 2024.
[3] Y. Meng et al., “Query-oriented two-stage attention-based model for code search,” J. Syst. Softw., vol. 207, Art. no. 1123436, Jan. 2024.
[4] Y. Meng et al., “Two-stage attention-based model for code search (TabCS),” in Proc. IEEE SANER, 2023, pp. 213–223.
[5] X. Li et al., “A mutual embedded self-attention network for code search,”
[6] J. Syst. Softw., vol. 192, Art. no. 111591, Oct. 2022.
[7] V. Hellen Doorn et al., “CodeGRU: Context-aware deep learning with gated recurrent unit for source code modeling,” arXiv:1903.00884, 2019.
[8] T. Zimmermann et al., “Big code search: A bibliography,” ACM Comput. Surv., vol. 56, no. 2, Art. no. 45, pp. 1–49, Feb. 2024.
[9] S. Wang et al., “A systematic literature review on the use of deep learning in software engineering research,” Tech. Rep., 2021.
[10] Y. Gu et al., “PSCS: A path-based neural model for semantic code search,” arXiv:2008.03042, 2020.
[11] H. Husain et al., “CodeSearchNet challenge: Evaluating the state of semantic code search,” arXiv:1909.09436, 2019.
[12] W. Sun et al., “A survey of source code search: A 3-dimensional perspective,” ACM Trans. Softw. Eng. Methodol., 2024.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 06 June 2026

[13] Y. Liu et al., "Survey of code search based on deep learning," arXiv:2305.05959, 2023.
[14] M. Fang, L. Wang, and H. Hu, "An adaptive model for cross-domain code search," Inf. Softw. Technol., vol. 186, Art. no. 107827, Oct. 2025.
[15] M. Fang, L. Wang, and H. Hu, "An adaptive model for cross-domain code search," *Information and Software Technology*, vol. 186, Art. no. 107827, 2025.
[16] H. Husain *et al.*, "CodeSearchNet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.