Pinnacle Academic Press
Proceedings Series

Vol. 2 2025

Article **Open Access**

Research on Intelligent Code Search Technology Based on Deep Learning

Anyi Chen [1,*]

[1] Walmart E-commerce Search Team, Wal-Mart Associates Inc., Bentonville, Arkansas, 72716, USA
* Correspondence: Anyi Chen, Walmart E-commerce Search Team, Wal-Mart Associates Inc., Bentonville, Arkansas, 72716, USA

Received: 16 April 2025
Revised: 29 April 2025
Accepted: 10 May 2025
Published: 12 June 2025

Abstract: The application of code intelligent search technology is gradually becoming a key means to solve the bottleneck of development efficiency in the context of the continuous expansion of software development project scale and the increasing demand for code reuse. This study explores the core applications of deep learning in code search, focusing on semantic representation techniques for code, enhanced semantic matching using Transformer architectures, the use of graph neural networks for code structure parsing, the application of contrastive learning strategies for semantic correlation mining, and the performance of multimodal fusion models in integrating code with natural language descriptions. An innovative intelligent code search system has been developed for system architecture scenarios. It includes the refinement of deep neural network models, the design of retrieval processes based on a dual-tower architecture, and strategies for resource allocation and technology selection during model deployment. Through research, significant progress has been revealed in the complex semantic integration and efficient search of deep learning in code search, opening up new research directions for the development of intelligent code search systems.

Keywords: deep learning; code search; code semantics; intelligent search

1. Introduction

In the process of software development, as software scale and logical complexity continue to increase, programmers need to quickly locate and reuse code snippets. The conventional code search method relies on simple correspondence of keywords, which has shortcomings in grasping the deep meaning of the code and the search results are not accurate enough. Thanks to the rapid advancement of deep learning technology, the intelligent upgrade of code search has ushered in a new opportunity. Deep learning can extract the semantic and structural features of code from large datasets, offering robust theoretical foundations and technical support for developing intelligent code search systems. This article is based on the basic principles of deep learning, and conducts in-depth research on the core technology of code intelligent search. It proposes an efficient and scalable intelligent code search solution in model optimization and system architecture design. This research achievement has opened up new directions for the development of code search technology and improved the search intelligence experience for developers.

2. Basic Theories of Deep Learning

Deep learning technology is an advanced machine learning method based on artificial neural networks, which simulates the way the human brain processes information [1].

The core concept is to use multi-level neural networks to abstract and train information layer by layer, thereby automatically modeling and reasoning complex information. Compared to traditional machine learning methods, deep learning has superior representation capabilities and can extract advanced features from large amounts of data, especially when dealing with unstructured data such as images, audio, and text, demonstrating its unique superiority. The theoretical pillars of deep learning involve backpropagation, gradient descent, and the use of nonlinear activation functions. By solving the derivative of the loss function with respect to the network weights, the backpropagation method can automatically adjust various parameters of the model. The introduction of non-linear activation functions endows neural networks with the ability to simulate complex functions. Thanks to its massive computing power and datasets, deep learning has rapidly developed, promoting the widespread application of convolutional neural networks, recurrent neural networks, and self attention structures (i.e. Transformers).

3. Key Technologies of Deep Learning in Code Search

3.1. Code Semantic Representation

Deep learning techniques are crucial for semantic representation of code in code search. This method converts programming code into corresponding semantic vectors, enabling deep networks to more accurately grasp the intrinsic connections between codes, thereby improving the intelligence level of search. Common code semantic representation techniques include vocabulary embedding techniques, syntax tree embedding techniques, and graph neural network embedding techniques. For example, in an experiment conducted on the CodeSearchNet dataset, Python code snippets were selected and a BERT pre trained model was used to generate vector representations of the code. These representations were compared with the vector of the query statement for cosine similarity to find similar code snippets. Table 1 shows the Top-1 accuracy and average search time achieved by different characterization techniques on the test set [2].

Table 1. Data Analysis Table.

Embedding method	Top-1 accuracy (%)	Average search time (ms)
Word embedding	68.5	10.2
Grammar tree embedding	73.2	12.5
Image embedding	78.6	15.7
Pre trained model	82.1	9.8

Based on semantic representation through code embeddings, this method calculates the similarity between a code fragment $\vec{c}i$ and a query $\vec{q}$ using the following formula:

$$Sim(c_i,q) = \frac{\vec{ci}\cdot\vec{q}}{||\vec{c_i}||\,||\vec{q}||} \quad (1)$$

In formula (1), c_i and q are the embedding vectors of the code snippet and query text, respectively, generated by a deep learning model $\|\cdot\|$ representing the norm of a vector, $\cdot$ represents the dot product of the vector. The compatibility evaluation tool between this code snippet and the query statement. Research shows that adopting CodeBERT to implement vector embedding processing for code improves the accuracy of code search, accelerates search speed, and provides strong technical support for intelligent code search.

3.2. Semantic Matching Based on Transformer

In the field of intelligent code search, transformer architecture plays a crucial role, with its unique multi attention head design effectively grasping the deep semantic connections between code and query statements [3]. Based on the code semantic correspondence technology of Transformer, the code and natural language queries are transformed into embedded vectors, and then deep neural networks are used to match and analyze these vectors, achieving precise search results. In the testing phase, the CodeSearchNet

dataset was selected, and the CodeBERT, based on the Transformer architecture, was used to encode the code and queries into a unified representation. After model training, it is possible to estimate the semantic similarity between code snippets and query statements. We compared the performance differences of different models in terms of Top-1 accuracy and average search time through data tables (Table 2).

Table 2. Performance Comparison of Different Models.

Model	Top-1 accuracy (%)	Average matching time (ms)
BiLSTM	69.3	11.5
CNN	74.1	9.8
Transformer	82.5	8.7
CodeBERT	86.8	7.2

The semantic matching calculation formula based on Transformer is as follows:

$$Sim(c,q)=softmax(\frac{QK^T}{\sqrt{d_k}}) \quad (2)$$

In formula (2), $Q = W_q \cdot X_q$ is the query matrix of the query text, $K = W_k \cdot X_c$ is the key matrix of the code fragment, d_k is the dimension of the attention vector, and $softmax$ generates the score results of embedded matching by applying normalization strategy to the similarity distribution model and combining weighted accumulation method. Analysis of the table data and application of relevant formulas show that CodeBERT's Transformer architecture and multi-head attention mechanism significantly improve the accuracy of semantic matching and reduce matching time. These results demonstrate CodeBERT's effectiveness as an advanced technical solution for intelligent code search.

3.3. Application of Graph Neural Networks in Code Structure Analysis

Graph neural networks in intelligent code search primarily focus on analyzing the internal structure of code. The code syntax is complex and the semantic structure is also quite rich, such as abstract syntax trees, program control flowcharts, and data flow diagrams, all of which can be represented in the form of graphs. Graph neural networks can extract the overall and detailed features of code by transmitting and summarizing node information in the graph structure. In practical applications, Java code snippets are selected as datasets and transformed into abstract syntax tree forms. Graph neural networks are used to create structured embeddings rich in semantic information. The search performance of the code is evaluated by comparing the structured embeddings with the corresponding query statements. Table 3 shows the comparison data of different models in terms of accuracy and processing efficiency.

Table 3. Comparison of Different Models.

Model	Top-1 accuracy (%)	Average processing time (ms)
BiLSTM(Sequence Model)	70.2	15.6
CNN(Plane structure model)	75.8	12.3
Basic GNN (GCN)	80.5	9.7
Graph Attention Network (GAT)	84.6	8.9

The basic propagation formula of GNN is:

$$h_v^{(k+1)} = \sigma\left(W^k \cdot AGG\left(\left\{h_u^{(k)}: u \in \mathrm{N}(v)\right\}\right)\right) + b^{(k)} \quad (3)$$

In formula (3), $h_v^{k=1}$ is the embedding vector of node v at the k+1th layer; $N(v)$ represents the set of neighbors of node v; AGG is an aggregation function, such as summation or averaging; $W^{(k)}$ and $b^{(k)}$ are weight matrices and biases; σ is the activation function. Combining graph attention mechanism (GAT), the aggregation function can be further improved as follows:

$$\text{AGG}= \sum_{u\in N(v)} \alpha_{\text{vu}} \cdot h_u^{(k)}, \alpha_{\text{vu}} = \frac{exp(e_{\text{vu}})}{\sum_{j\in N(v)} exp(e_{\text{vj}})}, e_{\text{vu}} = a^T[W \cdot h_v || W \cdot h_u] \quad (4)$$

In formula (4), α_{vu} refers to the weight assigned to the attention level of node v towards its neighboring node u. Graph neural networks effectively extract structural features of code, improving both semantic matching accuracy and processing speed. This plays an indispensable role in the field of intelligent code search.

3.4. Comparative Learning

Contrastive learning, as an unsupervised or semi-supervised learning method, captures the semantic structure of the embedding space by comparing differences between samples, enhancing the model's understanding of the semantic associations between code and queries. In the field of intelligent code search, this method optimizes the semantic vector distribution of the embedding space by establishing positive and negative sample sets, minimizing the distance between similar samples while maximizing the distance between dissimilar ones (Table 4) [4].

Table 4. Data Comparison and Analysis Table.

Method	Positive sample similarity	Negative sample similarity	Tightness of embedding space	Time complexity
Euclidean distance comparison	0.82	0.35	medium	O (N^2)
Cosine distance comparison	0.89	0.29	tall	O (N^2)
Weighted cosine comparison	0.92	0.25	higher	O (N^2)
Self supervised contrastive learning method	0.94	0.21	highest	O (N)

The optimized contrastive learning strategy has achieved significant results in narrowing the distance between similar samples, effectively rejecting heterogeneous samples, improving the accuracy of semantic matching, and providing solid support for the core part of code search technology.

3.5. Multimodal Joint Modeling

The modeling method that integrates multimodal information plays a key role in the field of code intelligent search, integrating different forms of code representation such as textual description, internal structure, and dynamic behavior. Deep neural networks are used to integrate and jointly represent these heterogeneous data, enhancing the accuracy and robustness of code search. In this type of modeling technique, text patterns generally cover textual descriptions or annotations of code parts, structural patterns involve abstract syntax structures or control flowcharts of code, and behavioral patterns describe the functional properties or execution performance of code. By extracting effective features from multidimensional data and applying deep integration techniques, the semantic features of the code can be more comprehensively revealed. The following Table 5 shows the performance comparison between single mode and multi-mode modeling.

Table 5. Data Comparison and Analysis Table.

Model type	Unimodal	Multimodal (text + structure)	Multimodal (text + structure + behavior)
Top-1 Accuracy (%)	73.4	85.2	90.6
Model complexity	Low	In the middle	high
Data demand quantity	Low	In the middle	high

Generalization ability	In the middle	high	highest

The joint construction strategy that integrates multiple modalities enables deeper multimodal integration, improving semantic expressiveness and search accuracy in code retrieval tasks, improved the ability to express code semantics and search accuracy, and provided solid technical support for intelligent code search technology.

4. Design of Code Intelligent Search System Based on Deep Learning

4.1. Training and Optimization of Deep Learning Models

In building an intelligent code search system, the training of deep learning models is the core. This system adopts the Transformer framework and has undergone deep customization of the model based on the syntax characteristics of the programming language [5]. The dataset used is collected from numerous open-source code repositories, covering code examples from mainstream programming languages such as Python, Java, C++, and more. In order to improve the model's grasp of code meaning, structured code annotation data was utilized, and a series of data augmentation techniques were implemented, such as extracting code annotations and replacing variable names. During the optimization phase of the model, methods such as learning rate adjustment and gradient pruning were employed to maintain robustness when training on large amounts of data. The loss function design of the system adopts weighted cross entropy that integrates code semantic distance to optimize model performance. The formula is as follows:

$$L = -\frac{1}{N}\sum_{i=1} w_i \cdot log(p_i) + \lambda \cdot d(s_i, t_i) \quad (5)$$

In formula (5), w_i represents the weight factor, p_i represents the prediction probability of the i-th sample, $d(s_i, t_i)$ refers to the semantic gap between codes, and λ is the adjustment factor. The following Table 6 shows the comparison of model performance for various training parameter settings:

Table 6. Model Performance Comparison Table.

Parameter configuration	Learning rate	Batch size	Model accuracy (Top-1)	Model accuracy (Top-5)	magnitude of the loss
Configuration A	1e-4	32	85.4%	96.3%	0.234
Configuration B (Optimize learning rate)	5e-5	32	87.6%	97.1%	0.198
Configuration C (Increase batch size)	5e-5	64	88.3%	97.5%	0.175
Configuration D (Comprehensive optimization)	3e-5	64	89.5%	98.1%	0.150

After comprehensive optimization, the configuration (Configuration D) has shown significant progress in numerous evaluation criteria, which leads to higher search accuracy and deeper semantic understanding to the code intelligent search system.

4.2. Search Process and Model Design

The design of the search mechanism for the code intelligent search system for deep learning should focus on the following key steps: input processing of user queries, extraction of feature vectors, application of deep neural network matching models, and output of final results. At the beginning, this process will convert the code snippets or natural language descriptions entered by the user into corresponding vector representations. Subsequently, a deep learning model is used to compare and analyze the semantic vectors of various code fragments in the code repository, identifying the code fragments most se-

mantically correlated with the user query. In this process, a dual tower network architecture with Transformer as the core was selected, which can map user input and code fragments to the feature space respectively, and use a common or exclusive weight system to evaluate similarity. The core formula is as follows:

$$S(\mathrm{u,c}) = \frac{\sum_{i=1}^{n} w_i \cdot (f_u(x_i) \cdot f_c(y_i))}{\sqrt{\sum_{i=1}^{n} f_u(x_i)^2} \cdot \sqrt{\sum_{i=1}^{n} f_c(y_i)^2}} \quad (6)$$

In formula (6), $S(\mathrm{u,c})$ represents the semantic similarity between user input u and code snippet c, $f_u(x_i)$ and $f_c(y_i)$ are personalized feature mapping operations, and w_i represents the weight coefficients corresponding to each feature. In order to explore the performance advantages and disadvantages of various models in semantic matching, this study compared correlation scores and computational efficiency (Table 7) [6].

Table 7. Comparison Results.

Model types	Average correlation score	Query time (ms)	Model parameter quantity (M)
Traditional model based on TF-IDF	0.62	45	—
Matching model Based on Word2Vec	0.73	56	12
Twin tower model based on Transformer	0.89	34	24

The data shows that the twin tower model using Transformer architecture has improved both relevance rating and query efficiency, and its optimization effect is more prominent in handling large code repository scenarios.

4.3. Model Deployment Environment and Technology Selection

In the process of implementing a deep learning based code intelligent search system, it is necessary to comprehensively measure the computational complexity, hardware resource requirements, and response speed requirements of the model, and find the optimal balance between performance and cost [7]. When selecting deployment scenarios, cloud architectures capable of handling high concurrency and providing low latency should be prioritized, and containerization technology should be utilized to ensure flexible scalability and continuous stable operation of the system. In terms of selecting technical solutions, it is recommended to use efficient computing support frameworks (such as TensorFlow Serving or TorchServe) to deploy model services, using message brokers (e.g., Kafka) and searchable storage engines (e.g., Elasticsearch) to work together to achieve effective processing and flow of data. The core resource allocation formula is as follows:

$$R(t) = \frac{\lambda}{\mu} + \beta \cdot \sigma^2 \quad (7)$$

In formula (7), $R(t)$ represents the resource consumption per unit time, λ and μ represent the request arrival rate and processing rate, β is the concurrency factor, and σ^2 represents the variance of the request load. The formula is used to dynamically adjust the number of model instances to ensure stable system performance (Table 8).

Table 8. Comparison Data of Different Deployment Schemes.

Deployment plan	Average response time (ms)	Throughput (requests/second)	Cost (monthly average, USD)
Single machine deployment	230	150	500
Cloud deployment (GPU optimization)	95	500	1200

Distributed deployment (containerization)	60	1200	1600

Statistical analysis indicates that integrating distributed architecture with containerization significantly reduces response latency and enhances overall system throughput can enhance the processing capability of the system while compressing response speed to the extreme, thereby making it an optimal solution for supporting large-scale and high-frequency code search operations. With the scalability of distributed systems, this architecture can flexibly respond to changing and complex query requirements as well as load fluctuations.

5. Conclusion

This study explores code intelligent search technology that integrates deep learning techniques. Based on a detailed analysis of the core technologies of semantic expression, matching mechanism, structural parsing, and multimodal integration of code, an efficient code search system architecture has been constructed. The paper also explains the training strategy, model optimization techniques, and deployment methods in detail. Compared to traditional baseline methods, the system demonstrates significant improvements in both semantic search accuracy and operational efficiency, successfully overcoming the challenges encountered in semantic parsing and multimodal information fusion during code search. The continuous evolution of deep learning technology, combined with automated tool processes and cloud computing resources, is expected to bring a broader horizon for the widespread application of intelligent code search. Research has provided developers with more advanced search tools and opened up new avenues for the practical application of deep learning in the field of software development.

References

1. O. F. Isife, D. Y. Dodo, J. A. Daramola, O. O. Abayomi, A. J. Afolabi, and R. O. Samuel, et al., "Development of a malicious network traffic intrusion detection system using deep learning,"*Int. J. Saf. Secur. Eng.*, vol. 13, no. 4, pp. –, 2023, doi: 10.18280/ijsse.130401.
2. L. Shen, C. Zhang, Y. Hu, Y. Wang, H. Lin, and Y. Zhao, et al., "Hybrid approach combining modified gravity model and deep learning for short-term forecasting of metro transit passenger flows,"*Transp. Res. Rec.*, vol. 2675, no. 1, pp. 25–38, 2021, doi: 10.1177/0361198120968823.
3. Y. Zhang, K. Guan, Z. Liu, C. Wang, T. Gui, and H. Yu, et al., "DeepWiPHY: Deep learning-based receiver design and dataset for IEEE 802.11 ax systems,"*IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1596–1611, 2020, doi: 10.1109/TWC.2020.3034610.
4. J. Ding and N.-W. Zheng, "RGB-D depth-sensor-based hand gesture recognition using deep learning of depth images with shadow effect removal for smart gesture communication,"*Sens. Mater.*, vol. 34, 2022.
5. M. Ishizaki, Y. Nakatani, and S. Nishida, "Developing a water quality estimation model by integrating deep learning with nonlinear time-series analysis,"*J. JSCE*, vol. 10, no. 1, pp. 288–306, 2022, doi: 10.2208/journalofjsce.10.1_288.
6. Y. Zhang, Z. Liu, X. Wang, J. Chen, Y. Zhou, and L. Sun, et al., "Damage detection of a pressure vessel with smart sensing and deep learning,"*IFAC-Pap. Online*, vol. 56, no. 3, pp. 379–384, 2023, doi: 10.1016/j.ifacol.2023.12.053.
7. L. Li, X. Huang, Q. Chen, T. Zhao, Y. Feng, and H. Zhang, et al., "On experiments of a novel unsupervised deep learning based rotor balancing method,"*Meas. Control*, vol. 55, no. 7–8, pp. 729–737, 2022, doi: 10.1177/00202940221115744.