International Journal of Knowledge and Language Processing
Volume 12, Number 2, Dec **2021**

AKLP International ©2021 ISSN 2191-2734
pp. **51—62**

Research on Business Process-oriented Code Parsing and Reuse Technologies

Yifeng Chen and Xin Tong

Peking University
No.5, Yiheyuan Road, Haidian District, Beijing, China
2001210207@stu.pku.edu.cn

Received October 2021; revised October 2021

ABSTRACT. *An automated machine learning system also referred to as automated ML or AutoML, is the process of automating the tedious tasks of machine learning model development like adjusting hyperparameters to improve development efficiency. Code parsing and reuse technology is the core of AutoML. This paper conducts an overview on the underlying technologies such as code reuse, code understanding, code representation, and code retrieval, thus summarizing both the challenges and future research directions of business process-oriented code parsing and reuse technologies. The study shows that the current code structure parsing and reuse technologies cannot meet the requirements of business process-oriented needs, since the existing algorithm platforms have their limitations, such as inflexible model calls and inadequate code self-assembly choices. Future research needs to focus more on the implicit structure and data flow of code, the function calls, and the standardization of data between algorithm components, and to model the dependency between algorithm components, to improve the accuracy and efficiency of code reuse.*

Keywords: Natural language processing, Code parsing, Code representation, Code resource structuring, Automated machine learning algorithm platform

1. **Introduction**

Nowadays, machine learning has important applications in the fields of automatic speech recognition, self-driving, intelligent retrieval, question answering systems, intelligent documents, automatic presentations, etc. This relies on the construction of well-performing machine learning pipelines, and traditional construction methods require both profound data scientists with statistical knowledge and domain experts with long-term experience in specific fields to build corresponding machine learning algorithms [1] . This process often requires multiple rounds of iterations. In addition, there is no one-size-fits-all algorithm [2] , and the algorithm often needs to be adjusted and improved for specific tasks to better adapt to actual business processes.

After automated machine learning is proposed, the creation of the whole process of

machine learning can be automated, including data use, feature selection, model design, and optimization methods. In this context, it is particularly necessary to study the related technologies of code self-assembly. By discussing code understanding, representation, reuse, and retrieval technology, the study can lay a foundation for realizing code self-assembly and promoting the further development of automated machine learning platforms from a perspective of a business process.

2. Related Technologies for Business Process-oriented Code Analysis and Assembly

Based on its structural and semantic features, the code in the repository is retrieved according to specific business requirements, which relies on the understanding of code structure, and algorithm components are assembled to generate a matching algorithm process that meets the business requirements. The essence of this process is to build a corresponding pipeline to complete automated machine learning. In this study, generation refers to the reuse of algorithmic code. The basis for reusing algorithmic code is the ability to disassemble the complete code structure, which is based on the understanding of the algorithm.

2.1 Research on Code Reuse Technology

Code reuse is a common research topic in the field of software engineering. With the development of the Internet, code hosting sites led by GitHub have accumulated a large amount of open-source software code, as well as a wealth of code in internal environments such as enterprises. Although the specific projects may be different, in the software development process, developers often encounter repetitive problems. This feature provides the possibility for code reuse. Code reuse [3] is commonly applied in enterprises. Developers implement code reuse in multiple forms for considerations of efficiency, cost, and code accuracy. A study shows that appropriate code reuse can quickly integrate functions and improve code accuracy. In short, code reuse can improve efficiency and reduce development costs when developing time and resource support are both limited.

The research on code reuse at home and abroad has gone through three stages, including the traditional code reuse stage, data mining−based reuse stage, and deep−learning−based code reuse stage.

(1) Traditional Code Reuse Stage

In the traditional code reuse stage, developers need to manually obtain reusable code from the Internet. In addition to source code, developers also need to pay attention to information such as code documents, design documents, code use cases, etc., to achieve code reuse accuracy. After obtaining the corresponding code, developers also need to make manual changes to the code to integrate with the existing project. This method effectively improves the work efficiency of software developers in the initial stage, but the reuse efficiency of code cannot meet the market’s needs due to excessive dependence on the developer's experience, high cost of obtaining code, and time-consuming tasks of query and confirmation.

(2) Code Reuse Stage Based on Data Mining

With the further development of the Internet, more and more developers choose to upload their own code to code hosting websites. At the same time, a large number of questions have been accumulated on question and answer websites like Stack Overflow. In software development processes, it is inevitable to encounter repetitive problems, faced with this situation, the traditional code reuse method will lead to inefficiency and problems such as defects caused by an incomplete understanding of code structure during reuse. To cope with this problem, code reuse technology based on big data mining came into being.

The code reuse technology based on big data mining can be divided into two dimensions, of which one is the reuse considered from the code fragment level. In this level of reuse process, code clone detection technology is used to detect whether there are similar copies in the project library, through finding similar copies, analyzing their differences, and extracting code modules and contained relationships for use in code reuse recommendation. Lin et al. combined the cloning detection result with difference comparison technology to detect the differences between instances, and at the same time believed that the differences of codes can be transformed into code variable points [4] . On this basis, the method analyzes the information of code change history and context, and fully explores differences among codes or their interrelationship, to provide developers with code reuse recommendations.

The second is the reuse of code function modules. The reuse of function modules is to extract designs and templates with similar functions from similar code blocks. Developers can implement specific functions for targeted needs on this basis[5] . This method requires users firstly to abstract the design model of the complete project from source code, including the class, APIs, methods, attributes, and other information in the project. Secondly, users need to determine the matching method of elements in the model, and based on this, gather the matched elements into multiple sets. The last step is to abstract based on multiple inter-set relationships and intra-set relationships to form template elements for programs, and then the code is generated.

(3) Code reuse based on deep learning

The rapid development of deep learning in the field of natural language processing has boosted its application in the field of code reuse recommendation. Deep learning focuses on implicit learning methods to achieve the purpose of reuse. At present, the code reuse recommendation method based on deep learning is mainly used in the training and prediction stage[6] . The code recommendation method based on traditional statistical learning [7-12] and deep learning [13-15] regards code as a sequence and applies traditional statistical models like N-gram or deep learning models like LSTM for learning and training. However, these research methods lack the consideration of the structural information of code, which makes it impossible to effectively capture the relationship between long-distance codes, and at the same time, they do not instantiate the parameters in code. To improve this situation, DeepAPIRec, a recommendation method based on Tree-LSTM, was developed[16] , which predicts abstract code sentences by training sentence models and constructing parameter models to realize the instantiation of code.

2.2 Code Comprehension Technology Research

The foundation of code reuse is code understanding and representation. Code is the encoding of knowledge, and code understanding is learning from code the domain knowledge. According to constructive learning theory, the process of learning and cognition is goal-oriented or model-driven inductive abstraction[17] . Code comprehension techniques are usually used for two types of tasks: revealing program semantics or optimizing programs. This study mainly focuses on code comprehension techniques to achieve the understanding and representation of program semantics.

To achieve code understanding, the research community usually employs reinforcement learning and stochastic compilation for hyper-optimization[18-19] , or borrows concepts from natural language processing (NLP) to process manually written code. The theoretical origin of this approach is that software corpora have statistical properties similar to natural language corpora, and these properties can be used to build better software engineering tools [20] . In the contextual representation of code tokens, one of the dominant approaches relies on lexicographical locations [13] ; while another approach mines the structure of code using data flow graphs [21] , control flow graphs[22] [24] , abstract syntax trees (ASTs) [25] , paths in ASTs [26] , or extended ASTs.

Algorithmic code contains strictly structural information, and as comprehensible natural languages, possesses semantic information. Adequate representation of the structural and semantic information of code is the basis for code parsing. Traditional approaches represented by Information Retrieval (IR) usually treat code pieces as natural language texts and model them based on tags. Kamiya and Kusumoto et al [27] and Sajnani and Saini et al [28] express codes in terms of sequences of tags or a series of tags, and apply them to code cloning detection. In addition, latent semantic indexing (LSI, Latent Semantic Indexing) [29] and document topic generation model (LDA, Latent Dirichlet Allocation) [30] have also been applied to analyzing source code which has yielded some results[31-32] . However, the problem with all these approaches is that they ignore the fact that code has richer and more explicit structural information and cannot be treated as purely natural languages or based on code characters (token) [33] .

White and Tufano[34] et al. and Wei and Li [35] showed that the contribution of syntactic knowledge in source code modeling is much better than the results of traditional token-based approaches. These approaches employ abstract syntax trees to represent code and achieve effective results. The abstract language description framework represents code fragments as trees with typed nodes [36] , where primitive structures correspond to atomic values including integers, identifiers, etc., and their nodes consist of both primitive types and related values. Compound types contain linguistic structures such as expressions, statements, etc., which contain constructor functions to specify the linguistic structures of nodes of that type. AST, the abstract syntax tree (abstract syntax tree) refers to a tree structure used to express the abstract syntactic structure of source code [37] , which is widely used in programming languages and software engineering tools. Zettlemoyer and Collins et al [38] view AST as a lightweight version of CCG, using abstract syntax trees to depict code structure.

2.3 Code Representation Technology Research

The basis for computing and processing algorithmic codes using machine learning methods is the vectorized representation of code. The vectorized representation of code requires consideration of sequential, structural, and semantic features of code. In this section, we study the vectorized representation of code based on the parsing and structuring of algorithms in terms of the sequential, structural, and semantic features of code.

In traditional code search engines and extractive code generation, the representation of code is usually done in the following ways.

1. treating all the code as natural language text and representing it. This approach can effectively use the semantic information contained in code, but introduces more noise and cannot exploit the structure contained in code.

2. Parsing code as an abstract syntax tree and representing it as a tree. This approach differs from treating code as a natural language text or as a tree structure and performing subsequent tasks such as similarity determination based only on its structure because a large amount of semantic information is lost in this case.

3. Representing code as a collection of consecutive method calls. The function naming best reflects the semantic information of a code component, so this approach can effectively represent the semantic information and also filter out some of the noise contained in code (e.g., irregular variable naming in code body, etc.). In addition, the set of method calls can, to a certain extent, represent the a priori relationships between components in the source code. However, this approach also ignores structural information, such as loop and reference structures contained in code.

The keyword-based representation approach only considers the sequential features of code and the natural language semantic information contained in the variables but does not model the structural features of code. To represent code more effectively, it is necessary to model both the structure and the semantic information of code. A more effective approach to model and represent structures is graph neural network-based representation learning. As a type of representation learning algorithm, network representation learning, is also referred to as graph representation learning. Network representation learning represents nodes in a network as low-dimensional dense vectors, which can maximize the preservation of inter-node information and has good applications in tasks such as node classification, link prediction, and community discovery[40] .

Network representation learning algorithms can be specifically classified into the following categories: 1) Matrix feature-based vector computation is an earlier algorithm used for network representation learning, where Tang and Liu[41] introduced modularity into loss function and eventually transformed the optimization problem into a feature vector computation problem for a certain relational matrix. 2) Simple neural network-based network representation algorithms are faster than the general approach and can achieve better performance. Its representative algorithm, the DeepWalk algorithm[42] is the first to introduce the techniques in deep learning into the field of network representation learning, fully applying the information of random walks in the network structure. 3) Matrix

decomposition-based methods, the core of which is to achieve the effect of dimensionality reduction by decomposing the relational matrix to obtain the node information. The GraRep algorithm[43] is one representative algorithm of this kind, which decomposes the relational matrix by SVD to obtain a network representation, and forms node representation with higher dimensionality and stronger ideology. 4) Methods based on deep neural networks, represented by SDNE [44] . 5) Community discovery algorithms, based on BIGCLAM [45] , a coverable community detection algorithm that learns a k-dimensional vector representation for each node in the network. 6) Network representation maintaining special properties, whose representative—HOPE algorithm[46] depicts each node as two different representations, and uses the JDGSVD algorithm for dimensionality reduction of the matrix.

2.4 Research on Code Retrieval Technology

The simplest method to search in code resource libraries is to use the tokenizer to extract all the character tokens from source code, to index the data based on these characters, and to use them as keywords to search. This method can perform cross-language code retrieval and work well for codebases containing multiple language sources, but this method cannot make good use of the semantic knowledge contained in code variables. Besides, it cannot explore the unique grammatical knowledge in the code of various programming languages. And it is impossible to determine whether the retrieved characters come from the method name or the variable. Sachdev et al. [39] use the AST information of code to extract method name, method call, enumeration, string text, and comment to form the document of code pieces, and represent the document with trained word vectors. They also calculate the similarity between the corresponding document representation and the representation of the language to be queried to retrieve code from a large-scale code base, and achieve better performance. Wehr [48] et al. used Siamese Neural Network to model the semantic similarity of source code to obtain the semantic representation of code. Gu et al. [49] used CODEnn to embed code and the natural language description of code together to create a semantic representation of Java source code. CODEnn uses a recurrent neural network (RNN) to model character token in API call sequence and method name, uses a multilayer perceptron to model the characters of code in non-API sequence, and fuse the output vectors of two models. By embedding the code and natural language together and adjusting the output vector, CODEnn gives a natural language description of the code. However, code comments are often context-sensitive, so comments may lack key information about code semantics. Dam et al. [50] used LSTM to model AST. The model was trained in an unsupervised manner. The training task was to predict the superior node based on the child nodes. Alon et al. [51] modeled the path in AST and evaluated the vector output of the model by predicting the method name in the code block to obtain the vector representation of code.

In the research based on code library retrieval and extraction of code to generate code use cases, MAPO[52] and UP-MINER[53] extract method call sequences from retrieved code fragments, mine high-frequency patterns in each category through clustering, and retrieve

required code use cases. GroupMiner [54] uses a graph-based approach to mine API use case models, and relies on frequent pattern mining for use case discovery. However, this method may produce a lot of redundant results. The MUSE proposed by Moreno et al. [55] uses a technique based on repeated code detection to cluster code blocks and select use cases. However, because it cannot effectively use the abstract information of the source code, it may also cause redundant results. Kim et al. [56] expressed code fragments as AST element vectors, clustered and sorted them according to vector similarity, and selected code use cases from different clusters to construct an intelligent search engine for code. These methods simplify the code modeling and also simplify the source code into method call sequence and feature vector, which may lose part of code information, such as code control flow, variable dependencies, etc... So, the generated code fragments may not meet the accuracy requirement, and cannot be reused.

3 Automated Machine Learning Algorithm Platform Construction

Through automated machine learning, algorithm professionals can automate tedious tasks such as adjusting hyperparameters, thereby improving work efficiency. Likewise, domain experts can build machine learning algorithms that meet the characteristics of the specific field without relying on data scientists. Even if they know little about machine learning, non-professionals can also rely on automated machine learning tools to build algorithms suitable for their tasks to complete corresponding work. Therefore, algorithm platforms, as the realization carrier of automated machine learning, have received extensive attention in the industry in recent years. In fact, the exploration of automated machine learning algorithm platforms first began in the academic circle. The University of British Columbia launched Auto-Weka (2013), followed by the University of Freiburg's Auto-sklearn (2014), the University of Pennsylvania's TPOT (2015), and Auto-Keras of Texas A&M University[57] .

In the industry, Google Cloud AutoML is a mature automated machine learning system, covering three fields—vision, language, and structured data with products including AutoML Vision, AutoML Video Intelligence, AutoML Natural Language, AutoML Translation, and AutoML Tables[58] . Google AutoML supports transfer learning, model structure search, and hyperparameter search. Microsoft also launched its own machine learning algorithm platform—Azure Machine Learning, which completes the automated construction of machine learning algorithms through neural network architecture search, model selection, feature extraction, hyperparameter adjustment, model compression, and other steps [59] .

Facundo Santiago[60] compares four popular automated learning platforms—Google Cloud AutoML, Azure Machine Learning, Auto-Keras, and Auto-sklearn in his blog. In this article, we use a table to organize his ideas[60] as follows:

TABLE 1. Comparative analysis of automated machine learning systems

Approach	*Open-*	*Cloud-*	*Tasks*	*Techniques*	*Training*

	source	*based*	*supported*		*framework*
Google Cloud AutoML	*No*	*Yes*	*CNN, RNN, LSTM for classification*	*Reinforcement learning with gradient policy upgrade*	*TensorFlow*
Azure Machine Learning	*No*	*Yes*	*Classification, Regression, Time-series forecasting, Computer vision (preview)*	*Probabilistic Matrix Factorization + Bayesian optimization*	*sklearn*
Auto-Keras	*Yes*	*No*	*CNN, RNN, LSTM for classification*	*Efficient Neural Architecture Search with Network Morphism*	*Keras*
Auto-sklearn	*Yes*	*No*	*Classification, Regression*	*Bayesian optimization + automated ensemble construction*	*sklearn*

In China, both Alibaba and Baidu have launched similar algorithm platforms. As Alibaba Cloud's machine learning platform, PAI provides cloud-native machine learning, covering PAI-DSW interactive modeling, PAI-Studio drag-and-drop visual modeling, and PAI-DLC distributed from training to the entire process of online deployment of the PAI-EAS model [61] . Baidu EasyDL supports a one-stop AI development process including data management, data annotation, model training, and model deployment[62] . However, the current platforms have limited exploration depth of code self-assembly technology, insufficient precision in indexing and parsing code pieces, and insufficient diversity of code assembly results, which in turn leads to the inflexibility of model calls. Therefore, they cannot achieve satisfactory performance in algorithmic intelligent retrieval, algorithmic process self-organization, and automatic construction of the algorithmic pipeline in response to business needs.

4 Conclusions

Existing automated machine learning platforms can free users from tedious work such as parameter adjustment, thus improving work efficiency. At the same time, the use of algorithm analysis and combination technology can avoid a number of human errors. However, existing algorithm platforms all share the problem of inflexible model calls, and they cannot customize automated algorithm construction and model selection for various business processes. Therefore, future research needs to further decompose the algorithm to understand its internal structure, obtain its structural information, and obtain its semantic information through the construction of language models. In terms of code structure analysis, the AST structure is too complex and detailed; while the CFG-based structure is quite simple, which results in the loss of information in the unresolved process statement.

Therefore, in the future, the code structure can be further combined with the detailed data flow in AST and the logic control flow of CFG. The data flow of code and the function call relationship between algorithm components should be paid more attention to. Through modeling the dependencies between components, the hidden structure in code should be mined. On the premise of fully analyzing algorithm structure, future research needs to focus more on improving the efficiency and accuracy of algorithm configuration. In future work, further study should be conducted on the normalization of data between algorithm components and efficiency improvement of algorithm self-organization systems based on existing automated machine learning technologies including hyperparameter search.

REFERENCES

[1] Zöller M A, Huber M F. Survey on automated machine learning[J]. arXiv preprint arXiv:1904.12054, 2019, 9.

[2] Wolpert D H, Macready W G. No free lunch theorems for optimization[J]. IEEE transactions on evolutionary computation, 1997, 1(1): 67-82.

[3] Haefliger S, Von Krogh G, Spaeth S. Code reuse in open source software[J]. Management Science, 2008, 54(1): 180-193.

[4] Lin Y, Xing Z, Xue Y, et al. Detecting differences across multiple instances of code clones[C]//Proceedings of the 36th International Conference on Software Engineering. 2014: 164-174.

[5] Lin Y, Meng G, Xue Y, et al. Mining implicit design templates for actionable code reuse[C]//2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2017: 394-404.

[6] PENG X, CHEN C, LIN Y. Context-based intelligent recommendation for code reuse[J]. Big Data Research, 2021, 7(1): 37-47.

[7] Hindle A, Barr E T, Gabel M, et al. On the naturalness of software[J]. Communications of the ACM, 2016, 59(5): 122-131.

[8] Allamanis M, Sutton C. Mining source code repositories at massive scale using language modeling[C]//2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013: 207-216.

[9] Nguyen A T, Hilton M, Codoban M, et al. API code recommendation using statistical learning from fine-grained changes[C]//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016: 511-522.

[10] Tu Z, Su Z, Devanbu P. On the localness of software[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014: 269-280.

[11] Nguyen T T, Nguyen A T, Nguyen H A, et al. A statistical semantic language model for source code[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. 2013: 532-542.

[12] Raychev V, Vechev M, Yahav E. Code completion with statistical language models[C]//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2014: 419-428.

[13] Dam H K, Tran T, Pham T. A deep language model for software code[J]. arXiv preprint arXiv:1608.02715, 2016.
[14] White M, Vendome C, Linares-Vásquez M, et al. Toward deep learning software repositories[C]//2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, 2015: 334-345.
[15] Nguyen A T, Nguyen T D, Phan H D, et al. A deep neural network language model with contexts for source code[C]//2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2018: 323-334.
[16] Chen C, Peng X, Xing Z, et al. Holistic Combination of Structural and Textual Code Information for Context based API Recommendation[J]. arXiv preprint arXiv:2010.07514, 2020.
[17] Jin Z, Liu F, Li G. Program comprehension: Present and future, Ruan Jian Xue Bao Journal of Software,2019,30(1):110-126 (in Chinese). http://www.jos.org.cn/1000-9825/5643.htm
[18] Bunel R, Desmaison A, Kumar M P, et al. Learning to superoptimize programs[J]. arXiv preprint arXiv:1611.01787, 2016.
[19] Schkufza E, Sharma R, Aiken A. Stochastic superoptimization[J]. ACM SIGARCH Computer Architecture News, 2013, 41(1): 305-316.
[20] Allamanis M, Barr E T, Devanbu P, et al. A survey of machine learning for big code and naturalness[J]. ACM Computing Surveys (CSUR), 2018, 51(4): 1-37.
[21] Allamanis M, Brockschmidt M. Smartpaste: Learning to adapt source code[J]. arXiv preprint arXiv:1705.07867, 2017.
[22] Nobre R, Martins L G A, Cardoso J M P. A graph-based iterative compiler pass selection and phase ordering approach[J]. ACM SIGPLAN Notices, 2016, 51(5): 21-30.
[23] Park E, Cavazos J, Alvarez M A. Using graph-based program characterization for predictive modeling[C]//Proceedings of the Tenth International Symposium on Code Generation and Optimization. 2012: 196-206.
[24] Xu X, Liu C, Feng Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 363-376.
[25] Bielik P, Raychev V, Vechev M. PHOG: probabilistic model for code[C]//International Conference on Machine Learning. PMLR, 2016: 2933-2942.
[26] Alon U, Zilberstein M, Levy O, et al. A general path-based representation for predicting program properties[J]. ACM SIGPLAN Notices, 2018, 53(4): 404-419.
[27] Kamiya T, Kusumoto S, Inoue K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code[J]. IEEE Transactions on Software Engineering, 2002, 28(7): 654-670.
[28] Sajnani H, Saini V, Svajlenko J, et al. Sourcerercc: Scaling code clone detection to big-code[C]//Proceedings of the 38th International Conference on Software Engineering. 2016: 1157-1168.
[29] Deerwester S, Dumais S T, Furnas G W, et al. Indexing by latent semantic analysis[J]. Journal of the American society for information science, 1990, 41(6): 391-407.
[30] Blei D M, Ng A Y, Jordan M I. Latent dirichlet allocation[J]. the Journal of machine Learning research, 2003, 3: 993-1022.
[31] Tairas R, Gray J. An information retrieval process to aid in the analysis of code clones[J]. Empirical

Software Engineering, 2009, 14(1): 33-56.
[32] Liu Y, Poshyvanyk D, Ferenc R, et al. Modeling class cohesion as mixtures of latent topics[C]//2009 IEEE International Conference on Software Maintenance. IEEE, 2009: 233-242.
[33] Pane J F, Myers B A. Studying the language and structure in non-programmers' solutions to programming problems[J]. International Journal of Human-Computer Studies, 2001, 54(2): 237-264.
[34] White M, Tufano M, Vendome C, et al. Deep learning code fragments for code clone detection[C]//2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2016: 87-98.
[35] Wei H, Li M. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code[C]//IJCAI. 2017: 3034-3040.
[36] Wang D C, Appel A W, Korn J L, et al. The Zephyr Abstract Syntax Description Language[C]//DSL. 1997, 97: 17-17.
[37] Baxter I D, Yahin A, Moura L, et al. Clone detection using abstract syntax trees[C]//Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272). IEEE, 1998: 368-377.
[38] Zettlemoyer L, Collins M. Online learning of relaxed CCG grammars for parsing to logical form[C]//Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). 2007: 678-687.
[39] Sachdev S, Li H, Luan S, et al. Retrieval on source code: a neural code search[C]//Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. 2018: 31-41.
[40] Tu C C, Yang C, Liu Z Y, et al. Network representation learning: an overview (in Chinese). Sci Sin Inform, 2017, 47: 980–996, doi: 10.1360/N112017-00145
[41] Tang L, Liu H. Leveraging social media networks for classification. [J]. Data Mining and. Knowledge Discovery, 2011, 23(3):447-478.
[42] Perozzi B, AI-Rfou R, Skiena S. DeepWalk: Online learning of social representations. [J]. KDD '14: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, 8: 701−710
[43] Cao S, Lu W, Xu Q. Grarep: learning graph representations with global structural information. In: Proceedings of the 24th ACM International Conference on Information and Knowledge Management, Melbourne, 2015. 891−900
[44] Wang D, Cui P, Zhu W. Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD InternationalConference on Knowledge Discovery and Data Mining, San Francisco, 2016. 1225−1234
[45] Yang J, Leskovec J. Overlapping community detection at scale: a nonnegative matrix factorization approach. In: Proceedings of the 6th ACM International Conference on Web Search and Data Mining, Rome, 2013. 587−596
[46] Ou M, Cui P, Pei J, et al. Asymmetric transitivity preserving graph embedding. In: Proceedings of the 22nd ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, 2016. 1105−1114
[47] Dettmers T, Minervini P, Stenetorp P, et al. Convolutional 2d knowledge graph embeddings[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2018, 32(1).

[48] Wehr D, Fede H, Pence E, et al. Learning Semantic Vector Representations of Source Code via a Siamese Neural Network[J]. arXiv preprint arXiv:1904.11968, 2019.
[49] Gu X, Zhang H, Kim S. Deep code search[C]//2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018: 933-944.
[50] Dam H K, Pham T, Ng S W, et al. A deep tree-based model for software defect prediction[J]. arXiv preprint arXiv:1802.00921, 2018.
[51] Alon U, Zilberstein M, Levy O, et al. code2vec: Learning distributed representations of code[J]. Proceedings of the ACM on Programming Languages, 2019, 3(POPL): 1-29.
[52] Zhong H, Xie T, Zhang L, et al. MAPO: Mining and recommending API usage patterns[C]//European Conference on Object-Oriented Programming. Springer, Berlin, Heidelberg, 2009: 318-343.
[53] Wang J, Dang Y, Zhang H, et al. Mining succinct and high-coverage API usage patterns from source code[C]//2013 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013: 319-328.
[54] Nguyen T T, Nguyen H A, Pham N H, et al. Graph-based mining of multiple object usage patterns[C]//Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering. 2009: 383-392.
[55] Moreno L, Bavota G, Di Penta M, et al. How can I use this method? [C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 880-890.
[56] Kim J, Lee S, Hwang S, et al. Towards an intelligent code search engine[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2010, 24(1).
[57] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss and R. Farivar, "Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools," 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), 2019, pp. 1471-1479, doi: 10.1109/ICTAI.2019.00209.
[58] “Google cloud automl”. [online] Available: https://cloud.google.com/automl/ (October 18,2021)
[59] “Microsoft Azure machine learning”. [online] Available: https://docs.microsoft.com/zh-cn/azure/machine-learning/(October 18,2021)
[60] Facundo Santiago. “Auto is the new black — Google AutoML, Microsoft Automated ML, AutoKeras and auto-sklearn”. [online] Available: https://santiagof.medium.com/auto-is-the-new-black-google-automl-microsoft-automated-ml-autokeras-and-auto-sklearn-80d1d3c3005c/ (October 18,2021)
[61] “Aliyun PAI”. [online] Available: https://help.aliyun.com/product/30347.html (October 18,2021)
[62] “Baidu EasyDL”. [online] Available: https://ai.baidu.com/easydl/(October 18,2021)