# Leveraging Comment Retrieval for Code Summarization

Shifu Hou[1], Lingwei Chen[2], Mingxuan Ju[1], and Yanfang Ye[1](✉)

[1] University of Notre Dame, Notre Dame, IN 46556, USA
{shou,mju2,yye7}@nd.edu
[2] Wright State University, Dayton, OH 45435, USA
lingwei.chen@wright.edu

**Abstract.** Open-source code often suffers from mismatched or missing comments, leading to difficult code comprehension, and burdening software development and maintenance. In this paper, we design a novel code summarization model CodeFiD to address this laborious challenge. Inspired by retrieval-augmented methods for open-domain question answering, CodeFiD first retrieves a set of relevant comments from code collections for a given code, and then aggregates presentations of code and these comments to produce a natural language sentence that summarizes the code behaviors. Different from current code summarization works that focus on improving code representations, our model resorts to external knowledge to enhance code summarizing performance. Extensive experiments on public code collections demonstrate the effectiveness of CodeFiD by outperforming state-of-the-art counterparts across all programming languages.

**Keywords:** Code summarization · Comment retrieval · Heterogeneous graph neural network · Fusion-in-Decoder

## 1 Introduction

Software developers benefit from billions of lines of source code that reside in online repositories [13,30]. Due to social coding properties, code often suffers from comments being mismatched or missing [11,29]. This makes code comprehension more difficult, which could easily increase the burden of software development and maintenance [26]. Hence, correctly summarizing the code behaviors is important and useful. As it is very expensive to manually acquire high-quality summarization, automatic yet effective code summarization pipelines are needed to address this laborious challenge.

Automatic code summarization is a rapidly expanding research area. Retrieval approaches were first proposed as a practice to exploit code keywords and similarity [25,32], which are limited to code formulation and easily fail when identifiers and methods are poorly named. Inspired by natural machine translation (NMT) from natural language processing (NLP), sequence-to-sequence (seq2seq) models then came to the forefront that read in the code as a sequence of tokens and generate a natural language sentence as a sequence of words [8,19,27]. As source code written in formal programming languages is syntactically structured [2], seq2seq models have recently adapted to more advanced graph-to-sequence (graph2seq) models. They leverage code structure and context through abstract syntax tree or constituency parsing tree [18], to boost the effectiveness of NMT techniques on code summarization [3,9,17,34].

Though the seq2seq and graph2seq models provide successful principles to solve the ambiguities and expressiveness in both source code and natural language descriptions, their inputs are inherently self-contained and struggle to leverage any external knowledge. In other words, while attending to depict source code and learn higher-level code representations for summarization, this line of research rarely takes advantage of any other relevant supplementary contexts. Hence our goal here is to investigate how much code summarization can benefit from retrieving external resources.

Retrieval-augmented pipelines from other fields such as open-domain question answering explore a retriever-reader framework, where a set of relevant passages are retrieved to enhance the knowledge coverage for question answering [14,15]. Inspired by their huge success, some recent works [22,32] start to shift such retrieval-augmented paradigms to extract different external resources for code summarization, which, however, either fail to capture useful connections between code snippets using traditional Dense Passage Retrieval (DPR) [15], or lead to unsatisfying performance improvement by introducing noisy information from external resources.

To address these limitations, in this paper, we propose a novel model that resorts to passage-like contexts from the collected data for code summarization. More specifically, the extracted supporting contexts refer to available comments paired with source code in the large training data collections. We argue that these text comments that are analogous to passages may contain "evidence" to the source code. To this end, on top of the state-of-the-art reader Fusion-in-Decoder [14], we design a retriever-reader framework for code summarization, called CodeFiD, which is shown in Fig. 1. In our CodeFiD, a retriever selects top $k$ relevant comments for a given code using dense representations, where we deploy heterogeneous graph [7] and in-batch negatives training [15] to fully leverage cross-fertilization of source code and comments. Then an FiD reader takes the source code along with its retrieved comments as inputs and aggregates their presentations to produce the final code summary.

## 2 Notations and Problem Definition

**Code Summarization.** A given code is denoted as a token sequence $x = (x_1, x_2, \ldots, x_n)$. A code summarization model is based on encoder-decoder architecture [10], where the encoder maps the sequence of tokens to a sequence of representations $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)$; the decoder produces the output natural language sentence $y = (y_1, y_2, \ldots, y_m)$ by maximizing the conditional probability $p(y_1, y_2, \ldots, y_m \mid \mathbf{z})$, such that:

$$y^* = \underset{y}{\operatorname{argmax}} \sum_{t=1}^{m} \log p(y_t \mid y_{<t}, \mathbf{z}) \tag{1}$$

In this paper, instead of introducing syntactic structure to facilitate code representation learning, we rely on external knowledge with respect to relevant comments from collections to supplement code and boost its summarizing performance.

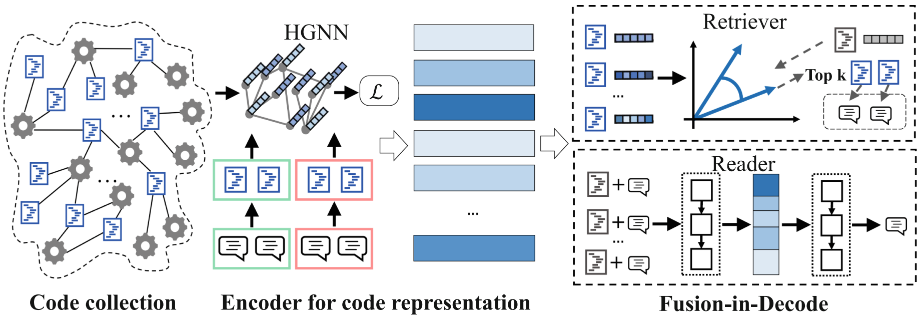


**Fig. 1.** In CodeFiD, a set of relevant comments are selected by HGNN-based retriever; then reader takes the code and retrieved comments to generate the summary.

**Comment Retrieval.** Given a code $x$ and a large set of comments $\mathcal{C}$, comment retrieval is to compute the similarity between $x$ and $\mathcal{C}$ using a similarity measuring function $f$ in order to retrieve $k$ $(k \geq 1)$ comments $c_k \in \mathcal{C}$ of which representation vectors are the closest to the code vector:

$$c_k = \begin{cases} \text{argmax}_{c \in \mathcal{C}} \ f(\mathbf{x}, \mathbf{c}) & k = 1 \\ \text{argmax}_{c \in \mathcal{C}, f(\mathbf{x},\mathbf{c}) < f(\mathbf{x},\mathbf{c}_{k-1})} \ f(\mathbf{x}, \mathbf{c}) & k > 1 \end{cases} \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{c} \in \mathbb{R}^d$ are representation vectors for the code $x$ and the comment $c$, respectively. In this paper, we define the similarity function $f$ between $x$ and $c$ using dot product of their vectors, which has been widely used in retrieval research [15].

## 3 Proposed Model

### 3.1 Retriever

For the retrieval of supporting comments, a typical way is to train encoders to jointly embed code and comments into unified vector space by minimizing a ranking loss with positive and negative $\langle x, c \rangle$ pairs as training instances [4,27]. Two code examples are given as follows, where code 1's comment is "*Parses the kml file and updates Google transit feed object with the extracted information*", and code 2's comment is "*Parses the given kml dom tree and updates Google transit feed object*". Though two blocks of code are very different, their comments are close to each other, as code 1 invokes an API defined by code 2 with some sharing identifiers. However, joint embedding paradigm may not be able to effectively catch such connections between code snippets.

```
def Parse(self, filename, feed): #code 1
    dom = minidom.parse(filename)
    self.ParseDom(dom, feed)


def ParseDom(self, dom, feed): #code 2
    shape_num = 0
    for n in dom.getElementsByTagName('Placemark'):
        p = self.ParsePlacemark(n)
        if p.IsPoint():
```

```
        self.stopNameRe.search(p.name)
    elif p.IsLine():
        self.ConvertPlacemarkToShape(p, feed)
```

To solve this issue, here we design a new yet more structured retriever to fully leverage cross-fertilization of code and comments: (1) the code data is first abstracted as a heterogeneous graph to model code interactions; (2) code representations are propagated and updated over this graph, which is completely guided by comment similarity, and then (3) top $k$ pieces of relevant training code are selected for the given code using the learned representations, where $k$ comments paired with these selected codes are finally retrieved to facilitate code summarization.

**Encoder Using Heterogeneous Graph.** As aforementioned, code pieces are related to each other through APIs and identifiers. Considering that code, APIs, and identifiers are of different types, we elaborate a heterogeneous graph (HG) [6,7,28,33] to represent the code data. To avoid introducing unexpected noises into graph, we intuitively extract those meaningful APIs and identifiers for HG construction. Specifically, the HG derived from code data collection is denoted as $G = (V, E, \mathbf{X})$, where $V$ is node set, $E$ is edge set to connect nodes when APIs/identifiers are included in code, and $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ is node feature matrix initialized using pretrained CodeBERT [8]. Through HG, it is easy to identify the relationships between any code pairs. Afterwards, we feed the resulting HG into a heterogeneous graph neural network (HGNN) $g_{\boldsymbol{\theta}}(\cdot)$ [31] to learn the higher-level code representations $\mathbf{Z} = g_{\boldsymbol{\theta}}(\mathbf{X}) \in \mathbb{R}^{|V| \times d'}$ that take advantage of heterogeneous neighborhood aggregation and code interactions, where $d'$ is the embedding size.

**Training.** Training HGNN encoder is a metric learning problem [16], such that the similarity between code representations can be a good ranking function. To achieve this goal, we need positive and negative code pairs to minimize the loss, which are unavailable explicitly at this stage. As we aim to back-propagate comment similarity to guide the updates on code representations, we design the following formulation: for each code $x$ and its comment $c$, any code from the training data whose pairing comments are $k$ nearest neighbors of $c$ is considered as a positive of $x$, and any code from the remaining is a negative of $x$. In this way, HGNN encoder can create a vector space such that similar comment pairs will enforce smaller distance between their code representations, while dissimilar comments will lead to large code representation discrepancy. To enable this positive and negative formulation, all comments are first mapped to embedding space using pretrained BERT [5] before fed to nearest neighbor searching.

Let $\mathcal{X} = \{\langle x_i, x_i^+, x_{i,1}^-, \ldots, x_{i,m}^- \rangle\}_{i=1}^n$ be the training data that consists of $n$ instances, where each instance includes one code snippet to summarize, one positive code snippet as well as $m$ negative code snippets. We can thus optimize the HGNN encoder by minimizing the following loss:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{x \in \mathcal{X}} \log \frac{e^{f(\mathbf{z}, \mathbf{z}^+)}}{e^{f(\mathbf{z}, \mathbf{z}^+)} + \sum_{j=1}^{m} e^{f(\mathbf{z}, \mathbf{z}_j^-)}} \tag{3}$$

Since we define $f(\cdot, \cdot)$ as dot product, we can use in-batch negatives [15] to reuse the computations and expedite the training in a more effective manner.

**Table 1.** The performance (BLEU-4) of different summarization pipelines on all datasets.

| Model | ALL | Ruby | JavaScript | Go | Python | Java | PHP |
|---|---|---|---|---|---|---|---|
| DistillCodeT5 | 20.01 | 15.75 | 16.42 | 20.21 | 20.59 | 20.51 | 26.58 |
| PolyglotCodeBERT | 19.06 | 14.75 | 15.80 | 18.77 | 18.71 | 20.11 | 26.23 |
| CoTexT | 18.55 | 14.02 | 14.96 | 18.86 | 19.73 | 19.06 | 24.68 |
| CodeBERT | 17.83 | 12.16 | 14.90 | 18.07 | 19.06 | 17.65 | 25.16 |
| Seq2Seq | 14.32 | 9.64 | 10.21 | 13.98 | 15.93 | 15.09 | 21.08 |
| RENCOS | 20.44 | 15.95 | 16.77 | 21.26 | 20.90 | 20.30 | 27.48 |
| REDCODER | 21.36 | 16.27 | 17.93 | 21.62 | 21.01 | **22.94** | 28.42 |
| CodeFiD (Ours) | **22.24** | **16.97** | **18.52** | **23.05** | **22.40** | 22.14 | **30.21** |
| w/ random retriver | 17.95 | 11.61 | 13.26 | 16.88 | 18.85 | 18.66 | 23.95 |
| w/ codebert retriver | 21.02 | 16.01 | 17.21 | 22.17 | 21.55 | 21.51 | 28.75 |
| w/o retriver | 18.25 | 13.82 | 14.35 | 18.42 | 19.35 | 19.10 | 24.08 |

**Comment Retrieval.** To retrieve relevant comments for a given code, we proceed with two steps based on code vectors output by HGNN encoder: (1) select $k$ code snippets whose representations are the closest to the given code in the same way defined in Sect. 2; and then (2) directly retrieve the pairing comments from these $k$ code snippets as augmentation to support code summarization.

### 3.2 Reader

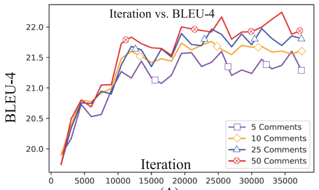
As this paper focuses on the investigation of the retrieval-augmented benefit for code summarization, we directly use FiD [14] to perform this task, which is based on a T5 model pretrained on unsupervised data [24]. More specifically, each retrieved comment is concatenated with the code, and then fed to the encoder independently from other comments to derive $k$ different embedding outputs. These outputs are all concatenated to be processed by the decoder using attention mechanism to generate the final code summary. Similar to open domain question answering implemented in FiD, though it is simple, this reader yields two significant advantages [14]: (1) scalable to large number of comments, and (2) effective to learn from multiple comments.

## 4 Experiments

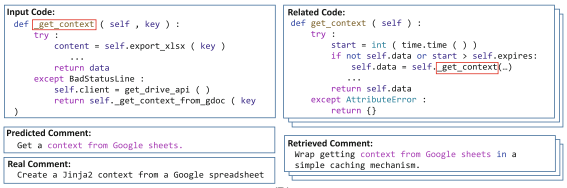
### 4.1 Experimental Setup

**Data.** We test our CodeFiD model on the CodeSearchNet dataset [12], which includes 908,224 training corpus, 44,689 validation corpus and 52,561 test corpus. This dataset has six programming languages, including Go, Java, JavaScript, PHP, Python and Ruby.

**Implementation Details.** We set for the number of retrieved comments per code as $k = 10$. We also evaluate its impact in Sect. 4.3. The parameter settings of HGNN and FiD are directly taken from [14,31]. All the experiments are performed under servers



(A)



**Input Code:**
```
def _get_context ( self , key ) :
    try :
        content = self.export_xlsx ( key )
           ...
        return data
    except BadStatusLine :
        self.client = get_drive_api ( )
        return self._get_context_from_gdoc ( key
)
```

**Predicted Comment:**
```
Get a context from Google sheets.
```

**Real Comment:**
```
Create a Jinja2 context from a Google spreadsheet
```

**Related Code:**
```
def get_context ( self ) :
    try :
        start = int ( time.time ( ) )
        if not self.data or start > self.expires:
            self.data = self._get_context(…)
           ...
        return self.data
    except AttributeError :
        return {}
```

**Retrieved Comment:**
```
Wrap getting context from Google sheets in a
simple caching mechanism.
```

(B)

**Fig. 2.** Performance of CodeFiD regarding the number of retrieved comments and a case study.

equipped with one RTX A6000 48GB GPU. As for software, we use the public repository of FiD$_{base}$[1] for reader, and DGL[2] for HGNN-based retriever.

**Evaluation Metrics.** We use BLEU-4 score [21] to measure the quality of generated code summaries, which calculates the similarity (i.e., cumulative 4-gram precision) between the generated sequence and reference sequence.

## 4.2 Comparisons with Baselines

We evaluate our proposed model CodeFiD by comparisons with recent code summarization models, including DistillCodeT5 [20], PolyglotCodeBERT [1], CoTexT [23], CodeBERT [8], Seq2Seq [20], and two retrieval-augmented models RENCOS [32] and REDCODER [22]. The results are reported in Table 1. We can observe that using retrieval yields significant performance gains. Despite using T5 network as encoder and decoder, CodeFiD enables retrieved comments augmented to code input to outperform existing state-of-the-art models. The best performing baselines are DistillCodeT5 (non-retrieval) and REDCODER (retrieval-augmented), where CodeFiD delivers an average improvement of 2.23 BLEU-4 score from DistillCodeT5 and further 0.88 BLEU-4 score from REDCODER across all programming languages.

## 4.3 Impact of Number of Retrieved Comments

We conduct the sensitivity analysis of how different choices of number of retrieved comments $k$ choices will affect the code summarization performance of CodeFiD. This evaluation is performed on single Python corpus. As illustrated in Fig. 2(A), when we enlarge $k$ from 5 to 50, the performance difference is trivial at lower steps, while the BLEU-4 score tends to rise to a higher level for larger retrieved comment number, especially at latter epochs. Considering that larger $k$ requires higher training computational budget, $k = 10$ seems a good trade-off between effectiveness and efficiency, whose average runtime for a batch (40 instances) costs 3.25 s.

[1] github.com/facebookresearch/FiD.
[2] www.dgl.ai.

### 4.4 Ablation Study

We also conduct ablation study to investigate the component contributions to CodeFiD performance. We formulate three alternative models, which are illustrated in Table 1. We can see HGNN retriever plays a crucial role in our model, which improves an average 3.99 BLUE score against the model without retriever. Random retriever underperforms by introducing irrelevant contexts that degrades code representations. CodeBERT retriever is promising, but fails to process the cases that rely on code interactions. Such a case is shown in Fig. 2(B), where CodeFiD benefits from structured retriever to locate the related code and retrieve its comment, which in turn provides the necessary evidence to produce the correct summary.

## 5 Conclusion

In this paper, we propose CodeFiD with a retriever-reader framework for code summarization. Specifically, our HGNN-based retriever selects a set of highly relevant comments, and then an FiD reader takes the source code along with its retrieved comments as inputs and aggregates their presentations to produce the final code summary. Extensive experiments on public code collections demonstrate the effectiveness of CodeFiD which outperforms state-of-the-art baselines. The improvement entailed by CodeFiD indicates that external knowledge, such as relevant comments from other code exploited in this paper, is beneficial for code summarization, which sheds light on a new direction for improving code summarization performance.

**Acknowledgments.** This work is partially supported by the NSF under grants IIS-2209814, IIS-2203262, IIS-2214376, IIS-2217239, OAC-2218762, CNS-2203261, CNS-2122631, CMMI-2146076, and the NIJ 2018-75-CX-0032. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

## References

1. Ahmed, T., Devanbu, P.: Multilingual training for software engineering. arXiv preprint arXiv:2112.02043 (2021)
2. Allamanis, M., Barr, E.T., Devanbu, P., Sutton, C.: A survey of machine learning for big code and naturalness. ACM Comput. Surv. (CSUR) **51**(4), 1–37 (2018)
3. Alon, U., Brody, S., Levy, O., Yahav, E.: code2seq: generating sequences from structured representations of code. In: International Conference on Learning Representations (ICLR) (2019)
4. Chen, L., Hou, S., Ye, Y., Xu, S.: Attributed heterogeneous information network embedding for code retrieval. In: Heterogeneous Information Network Analysis and Applications (2021)
5. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT (2019)
6. Fan, Y., Hou, S., Zhang, Y., Ye, Y., Abdulhayoglu, M.: Gotcha-sly malware! scorpion a metagraph2vec based malware detection system. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 253–262 (2018)

7. Fan, Y., Ju, M., Hou, S., Ye, Y., Wan, W., Wang, K., Mei, Y., Xiong, Q.: Heterogeneous temporal graph transformer: An intelligent system for evolving android malware detection. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. pp. 2831–2839 (2022)
8. Feng, Z., et al.: CodeBERT: a pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP, pp. 1536–1547 (2020)
9. Hellendoorn, V.J., Sutton, C., Singh, R., Maniatis, P., Bieber, D.: Global relational models of source code. In: International conference on learning representations (2019)
10. Hou, S., Chen, L., Ye, Y.: Summarizing source code from structure and context. In: 2022 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2022)
11. Hu, X., Li, G., Xia, X., Lo, D., Jin, Z.: Deep code comment generation. In: ICPC, pp. 200–210. IEEE (2018)
12. Husain, H., Wu, H.H., Gazit, T., Allamanis, M., Brockschmidt, M.: CodeSearchnet challenge: evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436 (2019)
13. Iyer, S., Konstas, I., Cheung, A., Zettlemoyer, L.: Summarizing source code using a neural attention model. In: ACL, pp. 2073–2083 (2016)
14. Izacard, G., Grave, E.: Leveraging passage retrieval with generative models for open domain question answering. In: Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pp. 874–880 (2021)
15. Karpukhin, V., et al.: Dense passage retrieval for open-domain question answering. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 6769–6781 (2020)
16. Kulis, B., et al.: Metric learning: a survey. Found. Trends® Mach. Learn. **5**(4), 287–364 (2013)
17. LeClair, A., Haque, S., Wu, L., McMillan, C.: Improved code summarization via a graph neural network. In: ICPC, pp. 184–195 (2020)
18. Ling, X., et al.: Deep graph matching and searching for semantic code retrieval. TKDD **15**(5), 1–21 (2021)
19. Loyola, P., Marrese-Taylor, E., Matsuo, Y.: A neural architecture for generating natural language descriptions from source code changes. arXiv preprint arXiv:1704.04856 (2017)
20. Lu, S., et al.: CodeXglue: a machine learning benchmark dataset for code understanding and generation. arXiv preprint arXiv:2102.04664 (2021)
21. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. In: Association for Computational Linguistics, pp. 311–318 (2002)
22. Parvez, M.R., Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K.W.: Retrieval augmented code generation and summarization. arXiv preprint arXiv:2108.11601 (2021)
23. Phan, L., et al.: Cotext: Multi-task learning with code-text transformer. arXiv preprint arXiv:2105.08645 (2021)
24. Raffel, C., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. **21**(140), 1–67 (2020)
25. Rodeghero, P., McMillan, C., McBurney, P.W., Bosch, N., D'Mello, S.: Improving automated source code summarization via an eye-tracking study of programmers. In: ICSE, pp. 390–401 (2014)
26. Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A.E., Li, S.: Measuring program comprehension: a large-scale field study with professionals. IEEE Trans. Softw. Eng. **44**(10), 951–976 (2017)
27. Yao, Z., Peddamail, J.R., Sun, H.: CoaCor: code annotation for code retrieval with reinforcement learning. In: The World Wide Web Conference, pp. 2203–2214 (2019)
28. Ye, Y., et al.: Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection. In: 28th International Joint Conference on Artificial Intelligence (IJCAI) (2019)

29. Ye, Y., et al.: ICSD: an automatic system for insecure code snippet detection in stack overflow over heterogeneous information network. In: Proceedings of the 34th Annual Computer Security Applications Conference, pp. 542–552 (2018)
30. Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S.: A survey on malware detection using data mining techniques. ACM Comput. Surv. (CSUR) **50**(3), 1–40 (2017)
31. Zhang, C., Song, D., Huang, C., Swami, A., Chawla, N.V.: Heterogeneous graph neural network. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 793–803 (2019)
32. Zhang, J., Wang, X., Zhang, H., Sun, H., Liu, X.: Retrieval-based neural source code summarization. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pp. 1385–1397. IEEE (2020)
33. Zhao, J., Wang, X., Shi, C., Hu, B., Song, G., Ye, Y.: Heterogeneous graph structure learning for graph neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 4697–4705 (2021)
34. Zügner, D., Kirschstein, T., Catasta, M., Leskovec, J., Günnemann, S.: Language-agnostic representation learning of source code from structure and context. arXiv preprint arXiv:2103.11318 (2021)