软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(6):2863−2879 [doi: 10.13328/j.cnki.jos.006910]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于关系图卷积网络的代码搜索方法*

周光有[1], 谢 琦[1], 余 啸[2]

[1](华中师范大学 计算机学院, 湖北 武汉 430079)

[2](武汉理工大学 计算机与人工智能学院, 湖北 武汉 430070)

通信作者: 周光有, E-mail: gyzhou@mail.ccnu.edu.cn

摘 要: 代码搜索是当下自然语言处理和软件工程交叉领域的一个重要分支. 开发高效的代码搜索算法能够显著提高代码重用的能力, 从而有效提高软件开发人员的工作效率. 代码搜索任务是以描述代码片段功能的自然语言作为输入, 在海量代码库中搜索得到相关代码片段的过程. 基于序列模型的代码搜索方法 DeepCS 虽然取得了很好的效果, 但这种方法不能捕捉代码的深层语义. 基于图嵌入的代码搜索方法 GraphSearchNet 能缓解这个问题, 但没有对代码与文本进行细粒度匹配, 也忽视了代码图和文本图的全局关系. 为了解决以上局限性, 提出基于关系图卷积网络的代码搜索方法, 对构建的文本图和代码图编码, 从节点层面对文本查询和代码片段进行细粒度匹配, 并应用神经张量网络捕捉它们的全局关系. 在两个公开数据集上的实验结果表明, 所提方法比先进的基线模型 DeepCS 和 GraphSearchNet 搜索精度更高.

关键词: 代码搜索; 关系图卷积网络; 细粒度匹配

中图法分类号: TP311

中文引用格式: 周光有, 谢琦, 余啸. 基于关系图卷积网络的代码搜索方法. 软件学报, 2024, 35(6): 2863–2879. http://www.jos.org.cn/1000-9825/6910.htm

英文引用格式: Zhou GY, Xie Q, Yu X. Code Search Method Based on Relational Graph Convolutional Network. Ruan Jian Xue Bao/Journal of Software, 2024, 35(6): 2863–2879 (in Chinese). http://www.jos.org.cn/1000-9825/6910.htm

Code Search Method Based on Relational Graph Convolutional Network

ZHOU Guang-You[1], XIE Qi[1], YU Xiao[2]

[1](School of Computer Science, Central China Normal University, Wuhan 430079, China)

[2](School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan 430070, China)

Abstract: Code search is an important research topic in natural language processing and software engineering. Developing efficient code search algorithms can significantly improve the code reuse and the working efficiency of software developers. The task of code search is to retrieve code fragments that meet the requirements from the massive code repository by taking the natural language describing the function of the code fragments as input. Although the sequence model-based code search method, namely DeepCS has achieved promising results, it cannot capture the deep semantics of the code. GraphSearchNet, a code search method based on graph embedding, can alleviate this problem, but it does not perform fine-grained matching on codes and texts and ignores the global relationship between code graphs and text graphs. To address the above limitations, this study proposes a code search method based on a relational graph convolutional network, which encodes the constructed text graphs and code graphs, performs fine-grained matching on text query and code fragments at the node level, and applies neural tensor networks to capture their global relationship. Experimental results on two public datasets show that the proposed method achieves higher search accuracy than state-of-the-art baseline models, namely DeepCS and GraphSearchNet.

Key words: code search; relational graph convolutional network; fine-grained matching

---

* 基金项目: 国家自然科学基金 (61972173); 中央高校自主科研经费 (CCNU22QN015); 武汉市知识创新专项基础研究项目 (20220108 01010278)

收稿时间: 2022-03-24; 修改时间: 2022-06-14, 2022-11-03; 采用时间: 2023-01-22; jos 在线出版时间: 2023-07-12

CNKI 网络首发时间: 2023-07-13

近年来, 代码仓库如 GitHub (https://www.github.com/)、Gitee (https://gitee.com/) 和 Bitbucket (https://www.bitbucket.org/) 中的开源项目呈井喷式增长, 优质的代码资源数量变得非常庞大, 如何准确高效地从代码仓库中搜索出符合程序员需求的代码, 是当前软件工程和自然语言处理交叉领域的热点研究方向之一. 与二进制代码搜索输入代码片段不同, 本文研究的目标是面向自由文本的代码搜索, 即输入一个描述代码片段功能的查询语句 (人类自然语言), 代码搜索任务是从大型代码库中找到最匹配的代码片段. 代码搜索工具不仅可以帮助程序开发人员找到常见程序功能的代码范例, 也可以帮助他们快速搜索其他开发者编写的个性化功能的优质代码片段, 这在很大程度上提高了程序开发人员的工作效率. 此外, 代码搜索技术现已成为程序合成、代码推荐与补全以及代码风格改善等程序构造技术的重要支撑[1]. 代码搜索的简要流程图如图 1 所示.

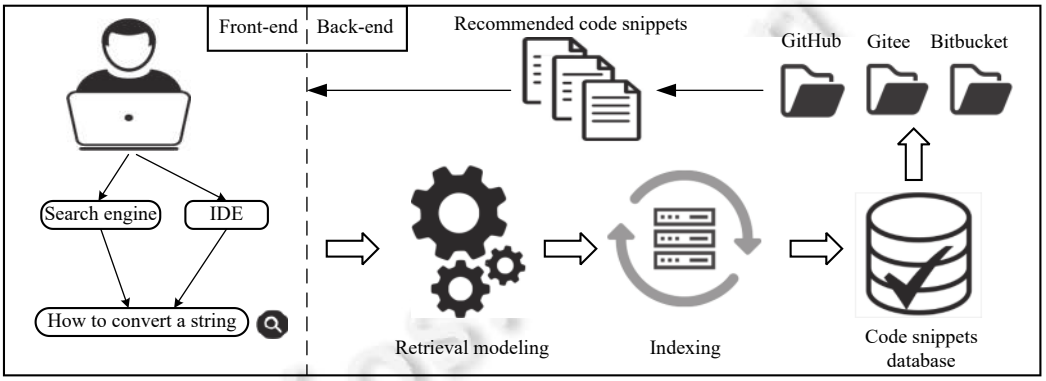

图 1 代码搜索的流程图

早期的研究大部分基于传统的信息检索技术. Bajracharya 等人[2]提出的 Sourcerer 以代码属性和其受欢迎程度作为评估指标, 结合 TF-IDF 统计方法返回推荐的代码片段. McMillan 等人[3]提出的代码搜索系统 Portfolio 基于关键字匹配得到程序员所需的功能性函数. Lu 等人[4]利用 WordNet[5]生成的自然语言查询同义词作扩展查询, 并对代码片段的方法签名进行关键字匹配. Lv 等人[6]提出的 CodeHow 算法考虑文本相似性和潜在 API 对代码搜索的影响, 并基于扩展查询和布尔模型搜索代码库. 黎宣等人[7]对代码片段基于提取的方法调用和结构特征作描述增强, 利用 Lucene 进行在线查询.

基于传统信息检索技术的代码搜索方法将查询语句和源代码均视为文本进行关键词匹配, 主要有两个缺点: 一是自然语言具有普遍的歧义性, 对其进行语义理解的准确性不高; 二是忽略了代码片段的深层结构特征. 为了解决上述问题, Gu 等人[8]首次将深度学习技术应用于代码搜索, 提出的 DeepCS 从方法名称、API 调用序列和源代码中包含的 token 信息 3 个角度表示源代码, 利用长短期记忆网络 (long-short term memory, LSTM) 提取相应特征, 丰富了代码层级的语义, 查询语句经过 RNN 编码模型得到相应嵌入. 相比于传统的 Sourcer[1]和 CodeHow[6], DeepCS 显著提升了搜索效果.

DeepCS 的提出启发了更多软件工程的研究者应用深度神经网络解决代码搜索这一任务. Yao 等人[9]提出了一种基于 sequence-to-sequence (Seq2Seq) 的代码注释生成工具 CoaCor, 可以生成用于代码搜索的良好注释. Chen 等人[10]提出了 BVAE 框架, 用两个变分自动编码器 C-VAE 和 L-VAE 分别表示代码片段和自然语言描述, 该方法旨在通过联合训练学习代码片段和自然语言描述的分布式向量表示. Wan 等人[11]提出了一个新的多模态注意力网络 MMAN, 并将其应用于代码搜索. 该网络由一个 LSTM, 一个 Tree-LSTM 和一个门控图神经网络组成, 分别用于代码的顺序标记, 代码的抽象语法树 AST 生成和控制流图的表示.

然而, 上述基于深度学习的方法只能捕捉到代码片段的浅层信息. 许多源程序可能存在各种长程依赖特征, 例如相同的标识符在源代码的多处进行关键性操作. 由于代码片段存在各种复杂的数据流, LSTM 等序列编码模型无法捕捉源代码的深层语义. 为了捕捉代码的深层结构信息, 学者们尝试用图数据表征代码片段的结构. 凌春阳等人[12]用 LINE 算法[13]嵌入代码图, 词袋模型嵌入语言查询, 并通过代码子图的搜索来匹配文本查询节点. 黄思远等人[14]基于重标签算法提取程序依赖图中的子图结构信息, 并基于 Doc2Vec 的思想将子图结构作为上下文信息得

到代码片段对应程序依赖图的向量嵌入表示.

Liu 等人[15]提出的 GraphSearchNet 模型是最新的基于图神经网络的代码搜索框架. GraphSearchNet 将源代码和查询文本分别编码成两个图, 用双向门控图神经网络来捕捉代码片段和查询文本的局部结构信息, 并通过多头注意力机制得到代码或文本的全局上下文信息. 实验结果表明 GraphSearchNet 能够有效刻画代码片段和自然语言描述的深层语义. 虽然基于图神经网络的 GraphSearchNet 模型取得了目前最好的结果, 但还存在以下不足.

(1) 代码片段和自然语言描述是两种截然不同模态的数据, 语法规则和语言结构的不同导致了语义映射的困难, 所以细粒度匹配对跨模态数据的搜索至关重要. GraphSearchNet 虽然通过双向门控图神经网络得到了代码和文本的图节点嵌入表示, 但没有对代码图和文本图的节点做任何交互匹配操作, 缺乏细粒度对齐的数据导致了他们的模型精度不够高.

(2) 全局视角下一对图的图级交互特征可能是学习图相似性的重要补充组件. GraphSearchNet 使用多头注意力机制聚合代码和文本的序列语义信息, 补充了代码图和文本图节点由于长程依赖忽视的上下文信息, 但仍然没有进一步对代码和文本的全局上下文信息进行比较和匹配, 忽视全局匹配的模型可能会因为丢失图级交互特征不能捕捉到代码和文本的全局关系.

为了解决上述不足, 本文提出了基于关系图卷积网络的代码搜索方法 GraphCS 模型. 首先构建了代码图和文本图, 并使用关系图卷积网络学习代码图和文本图的节点嵌入. 不同于 GraphSearchNet 在得到节点嵌入后就直接进行相似度度量, GraphCS 基于提出的多视角匹配函数, 对代码图和文本图的所有节点对进行跨图匹配对齐, 基于匹配后的特征更新节点嵌入表示, 其优点就是我们的模型能够捕捉到代码和文本的细粒度匹配关系, 有效解决了 GraphSearchNet 缺乏交互操作的第 1 个不足. 另外, 模型利用神经张量网络对代码图和文本图在多个维度上进行交互匹配, 能够捕捉到代码和文本的全局关系, 有效解决了 GraphSearchNet 忽略全局匹配关系的第 2 个不足. 在公开数据集 FB-Java 和 CSN-Python 上进行实验, 与两个先进的基线模型 DeepCS 和 GraphSearchNet 相比, GraphCS 在通用的搜索指标 SuccessRate@1/5/10 以及 MRR 上均有较大的提升.

1 问题定义

本文的研究目标面向自然语言输入的代码搜索, 是用文本搜索得到相应代码片段的过程. 本文从图数据可以有效表征结构信息的角度来定义代码搜索任务. 给定一个源代码语料库 P, 共有 $|P|$ 个代码片段, 代码搜索的目标是根据查询文本 q 从语料库中找到最匹配的代码片段 $\hat{p}$. 定义代码搜索任务的表述如下:

$$\hat{p}=\underset{p\in P}{\operatorname{argmax}}\, score(q,p)=\underset{p\in P}{\operatorname{argmax}}\, score(G_q,G_p) \tag{1}$$

这个代码搜索任务的核心是计算代码片段 p 和查询文本 q 之间的相似度分数 $score(p,q)$. 由于代码和文本都可以基于提出的图构建方法表示成图结构, 我们进一步将 $score(p,q)$ 转化为 $score(G_p,G_q)$. 其中 G_p、G_q 分别是代码和文本的图表示. 在本文中, G_p 和 G_q 都表示为有向和有标记的多关系图, 不同的边类型由有标记的边表示. 具体来说, 文本图 G_q 表示为 $(\mathcal{V}_q,\mathcal{E}_q,\mathcal{R}_q)$, 节点 $q_i\in\mathcal{V}_q$、边 $(q_i,r,q_j)\in\mathcal{E}_q$, 其中 $r\in\mathcal{R}_q$ 表示边的类型. 类似地, 代码图表示 G_q 为 $(\mathcal{V}_q,\mathcal{E}_q,\mathcal{R}_q)$, G_q 和 G_q 的节点数分别为 M 和 N. 为了描述方便, 在表 1 中给出了本文所用到的主要符号及其表示的含义.

2 GraphCS 模型介绍

本节详细介绍我们提出的方法, 具体来说, 基于关系图卷积网络的 GraphCS 模型由 3 个关键阶段组成.

• 首先为图构建和图嵌入阶段, 将查询语句基于选区解析树表示成文本图, 将代码片段基于增强关系表示的抽象语法树表示为代码图. 然后利用关系图卷积网络对代码图和文本图编码, 得到它们的初始节点嵌入表示.

• 核心阶段为语义匹配阶段, 语义匹配模块分为节点层面的匹配策略和图层面的匹配策略. 节点层面的匹配策略基于多视角匹配函数, 对文本图和代码图的所有节点进行跨图匹配操作以进行信息交互, 更新后的节点嵌入具

有更丰富的语义, 弥补了其他基线模型未对文本和代码进行细粒度匹配操作的不足. 图层面的匹配策略用于捕捉代码和文本的全局关系, 首先基于注意力机制得到代码图和文本图全局上下文感知的图嵌入表示, 然后利用神经张量网络得到代码图和文本图全局视角的相似性向量, 解决了其他基线模型没有进一步探索文本图和代码图全局语义信息的关系的问题.

表 1 符合及其含义

符号	含义或描述
G_p	代码图
G_q	文本图
$H_p=\{p_j\}_{j=1}^{N}\in\mathbb{R}^{(N,d)}$	代码图的初始节点嵌入表示, 嵌入维度为 d
$H_q=\{q_i\}_{i=1}^{M}\in\mathbb{R}^{(M,d)}$	文本图的初始节点嵌入表示, 嵌入维度为 d
f_m	多视角匹配函数
$H_p=\{\hat{p}_j\}_{j=1}^{N}\in\mathbb{R}^{(N,\tilde{d})}$	代码图的更新节点嵌入表示, 嵌入维度为 $\tilde{d}$
$H_q=\{\hat{q}_i\}_{i=1}^{M}\in\mathbb{R}^{(M,\tilde{d})}$	文本图的更新节点嵌入表示, 嵌入维度为 $\tilde{d}$
$h_p\in\mathbb{R}^d$	代码图全局上下文感知的嵌入表示
$h_q\in\mathbb{R}^d$	文本图全局上下文感知的嵌入表示
$NTN_{(h_p,h_q)}\in\mathbb{R}^k$	经过神经张量网络得到的相似性向量

• 最后是相似性搜索阶段, 根据基于多视角匹配函数的节点层面的匹配策略和基于神经张量网络的图层面的匹配策略分别得到的相似度分数, 计算一对代码图和文本图的最终相似度得分. 本文提出的 GraphCS 模型的整体模型框架如图 2 所示.

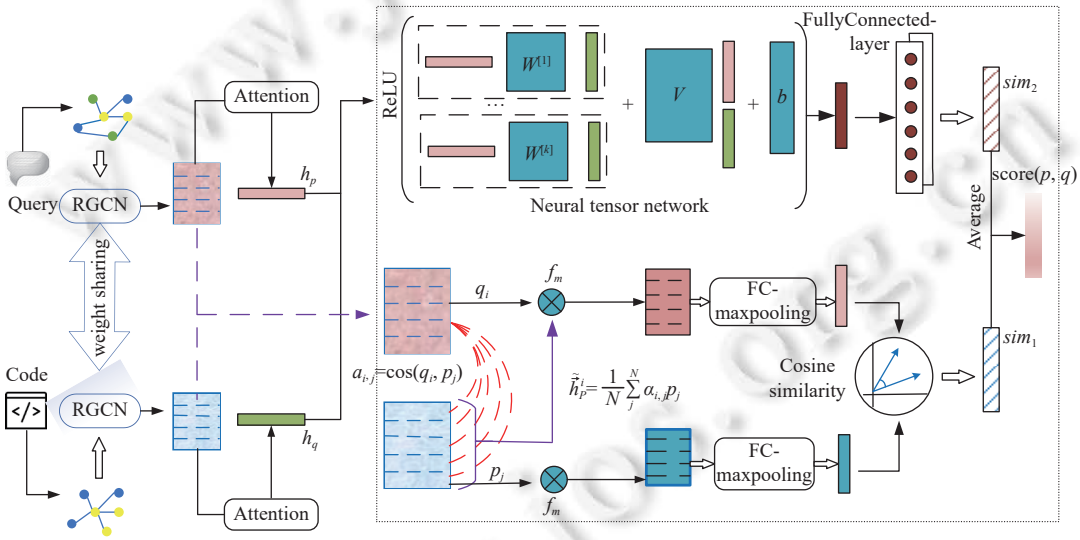

图 2 模型的整体框架图

2.1 图构建模块

本节详细介绍代码图和文本图是如何构建的. 如图 3 所示是一个 Java 语言编写的代码片段, 其函数功能是实现欧几里得算法, 计算得到两个整数的最大公约数. 这是一个可用于训练代码搜索任务的范例, 因为它包含了规范的代码实现和函数功能相应的注释文本.

2.1.1 代码图的构建

Allamanis 等人[16]提出的 Program Graph 是表示代码结构语义最流行的一种图构建方法, 基于此方法本文构

建了代码图. 代码图由语法节点、语法令牌和对应的边构成, 其骨干结构是代码片段的抽象语法树 (AST). 一个代码片段的抽象语法树由元组 $\langle N,T,X,s,\delta,\phi\rangle$ 表示, 其中 N 是一组非终端节点, T 是一组终端节点, X 是一组值, $s\in N$ 是根节点. 函数 $\delta: N\rightarrow (N\cup T)*$ 将非终端节点映射到对应的子节点列表, 其中, $*$ 表示 Kleene 算子, $(N\cup T)*$ 是由括号里的符号生成的一个无穷集合. 函数 $\phi: T\rightarrow X$ 将终端节点映射到与之相关的值. 而且除了根节点之外的每个节点在它的子节点列表中都只出现一次, 语法节点对应 $(N\cup T)$ 中的节点, 语法令牌对应终端节点的值.

```
/*
   Implements the euclidean algorithm
   find the greatest common divisior of a and b
*/
public static int          Greatest_common_divisor (int a, int b) {

    while (b != 0) {
        if   (a > b)
             a=a-b;
        else
             b=b-a;
    }
    return     a;
}
```

Natural language description

Corresponding code snippet

图 3　代码片段的实例

抽象语法树对代码片段进行词法分析和语法分析得到了有效表征代码语义的树结构, 但缺乏节点之间的数据流关系表达. 这里命名 3 种类型的边来增强 AST 的表示: Child 边用于连接 AST 中的所有语法节点, 在图中用黑色箭头表示; NextToken 边用于连接每个语法令牌和它的下一个语法令牌, 能够在图中保留程序片段的顺序序列信息, 在图中用蓝色实线表示; LastLexicalUse 边用于连接变量标识符和它最近一次在代码中的词汇用法, 在图中用绿色虚线表示. 对于实现欧几里得算法的代码片段, 图 4 展示了相应的代码图构建结果.

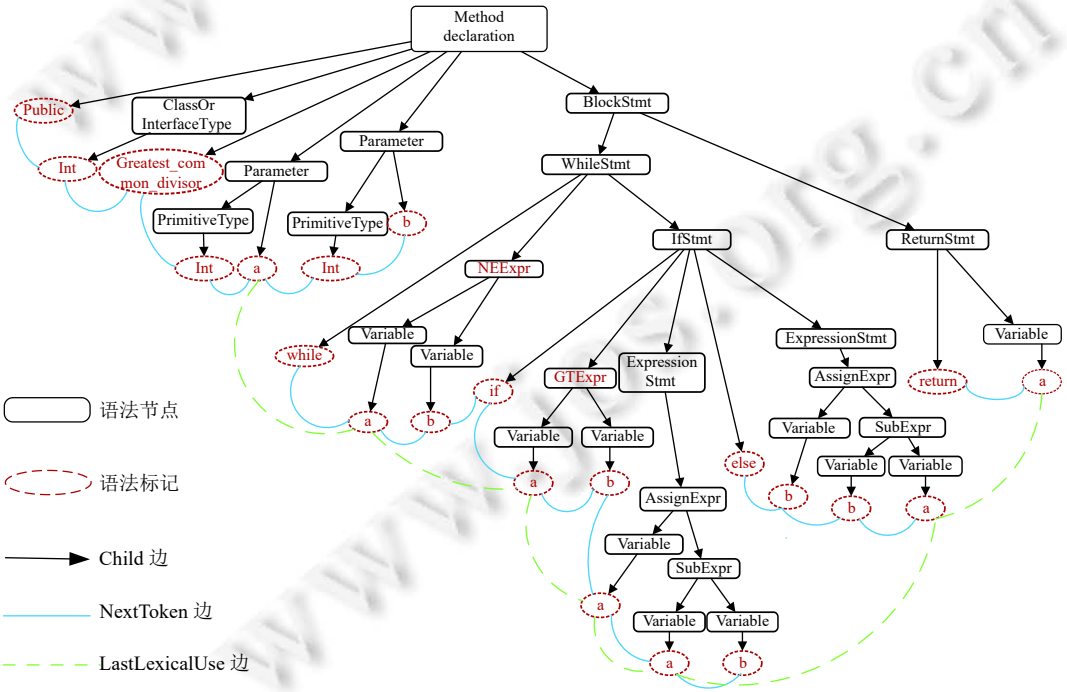

图 4　代码图的示例

2.1.2 文本图的构建

自然语言特定的顺序使得人们可以应用序列编码模型学习其语义, 但它们内部的结构信息也十分重要. 本文基于选区解析树[17]和词序特征来构建文本图, 与代码片段基于抽象语法树构建代码图有着类似的思想, 它们的表示都是高度分层的, 最后一层终端节点都是有意义的代码令牌或句子单词, 并且有明确的边类型关系, 相似的图结构表示使得模型更容易捕捉文本与代码片段的匹配关系.

选区解析根据上下文无关文法创建句法表示的树. 具体来说, 自然语言的上下文无关文法由四元组 (V,Σ,R,S) 表示. V 是非终端节点的集合, Σ 是与 V 不相交的一组终端节点. R 是一组文法规则 $V\to(V\cup\Sigma)*$, 将非终端节点映射到子节点, 子节点来自于无穷集合 $(V\cup\Sigma)*$. S 是开始变量, 用来表示整个句子. 另外, 为了表达文本的完整语义, 图构建时我们还考虑了文本的词序信息. 具体做法是将句子中的词 (即选区树中的终端节点) 链接起来, 这样可以捕捉到句子前向和后向的上下文信息. 对于实现欧几里得算法的代码片段, 其注释文本为“Implements the Euclidean algorithm”, 相应的文本图构造结果如图 5 所示.

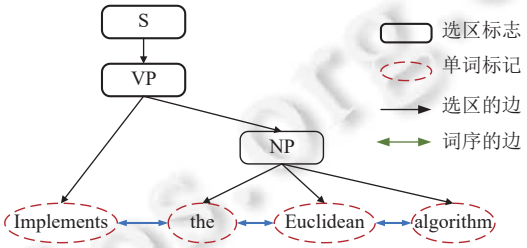

图 5 文本图的示例

2.2 图嵌入模块

图卷积网络[18]由于其对非欧几里得数据强大的建模能力受到了研究人员的大量关注. 关系图卷积网络 (RGCN)[19]是图卷积网络 (GCN) 在多关系型数据建模中的应用, 特别是在链接预测和实体分类任务中展现了良好的效果. 在图卷积网络中, 图中的每个节点 i 在第 $l+1$ 层的隐藏层表示如下:

$$h_i^{l+1}=\sigma\left(\sum_{j\in N_i}\frac{1}{c_i}W^{(l)}h_j^{(l)}\right) \tag{2}$$

其中, c_i 是归一化常数. RGCN 和 GCN 的主要区别在于, 在 RGCN 中, 边可以表示不同的关系. 在 GCN 中, 公式 (2) 中的权重 $W^{(l)}$ 由第 l 层中的所有边共享. 相比之下, 在 RGCN 中, 不同的边类型使用不同的权重, 并且只有相同关系类型的边与相同的投影权重 $W_r^{(l)}$ 相关联. 因此 RGCN 中第 $l+1$ 层实体的隐藏状态可以表示为公式 (3):

$$h_i^{l+1}=\sigma\left(W_0^{(l)}h_i^{(l)}+\sum_{r\in R}\sum_{j\in N_i^r}\frac{1}{\left|N_i^r\right|}W_r^{(l)}h_j^{(l)}\right) \tag{3}$$

其中, N_i^r 表示 $r\in R$ 关系下节点 i 的邻居索引集, $\left|N_i^r\right|$ 是归一化常数. 由于文本图和代码图都是有向图, 而且有多种类型的边关系. 因此采用关系图卷积网络来学习文本图和代码图的节点嵌入是非常合适的. 给定构建的文本图 $G_q=(\mathcal{V}_q,\mathcal{E}_q,\mathcal{R}_q)$, 关系图卷积网络 RGCN 定义图中每个节点 $q_i\in\mathcal{V}_q$ 的更新嵌入向量的传播模型如下:

$$q_i^{(l+1)}=ReLU\left(W_\Theta^{(l)}q_i^{(l)}+\sum_{r\in\mathcal{R}_q}\sum_{j\in N_i^r}\frac{1}{\left|N_i^r\right|}W_r^{(l)}q_j^{(l)}\right) \tag{4}$$

其中, $q_i^{(l+1)}$ 表示文本图的节点 q_i 在第 $(l+1)$ 层的更新嵌入向量, $\mathcal{R}_q$ 表示边类型的集合, $W_\Theta^{(l)}$ 和 $W_r^{(l)}$ 是 RGCN 模型的可学习参数, $ReLU$ 表示激活函数. 对于给定构建的代码图 $G_q=(\mathcal{V}_q,\mathcal{E}_q,\mathcal{R}_q)$, 同样使用上述 RGCN 模型对图中的每个节点邻域信息更新传播, 得到了文本图 G_q 的所有节点嵌入 $H_q=\{q_i\}_{i=1}^M\in\mathbb{R}^{(M,d)}$ 和代码图 G_p 的所有节点嵌入 $H_p=\{p_j\}_{j=1}^N\in\mathbb{R}^{(N,d)}$, 其中 M 和 N 分别代表文本图和代码图的节点个数, d 代表每个节点的维度.

2.3 基于多视角匹配函数的节点层面匹配策略

GraphSearchNet 的第 1 个不足是没有对代码图和文本图的节点做任何交互匹配操作, 为了得到代码图和文本图之间更细粒度的匹配关系, GraphCS 提出了基于多视角匹配函数的节点层面匹配策略. 为了捕捉文本图和代码图节点之间的细粒度语义匹配关系, 需要对它们进行跨图匹配操作. 跨图匹配是对图中的一个节点和另一个图的整体嵌入表示进行多视角匹配操作. 首先, 本文定义了一个多视角匹配函数 f_m 来计算两个输入向量 $\vec{x}_1$ 和 $\vec{x}_2$ 的相似性向量.

$$\widetilde{\vec{h}}[k]=f_m(\vec{x}_1,\vec{x}_2,\vec{w}_k)=\cos(\vec{x}_1\odot\vec{w}_k,\vec{x}_2\odot\vec{w}_k),\ k=1,\ldots,\tilde{d} \tag{5}$$

其中, $\odot$ 表示 element-wise 乘法操作, $\widetilde{\vec{h}}[k]\in\mathcal{R}$ 表示第 k 个视角的输出相似度得分, $\vec{w}_k\in\mathcal{R}^{d'}$ 表示表示第 k 个视角的可学习权重向量. 当考虑多视角匹配函数所有的 $\tilde{d}$ 个视角时, 可训练的权重矩阵将变为 $W_m=\{\vec{w}_k\}_{k=1}^{\tilde{d}}\in\mathcal{R}^{d'\times\tilde{d}}$. 输入向量 $\vec{x}_1$ 和 $\vec{x}_2$ 经过多视角匹配函数后就能得到 $\tilde{d}$ 维向量 $\widetilde{\vec{h}}\in\mathcal{R}^{\tilde{d}}$, 它衡量了两个输入向量的相关性. 值得注意的是, 所提出的 f_m 本质上与多头注意力机制[20]具有相似的精神. 二者的显著区别在于, 多头注意力机制使用 $\tilde{d}$ 个可训练权重矩阵, 而 f_m 使用 $\tilde{d}$ 个可训练权重向量代替, 也就是一个 $\tilde{d}$ 维权重矩阵. 本文的方法使用更少的训练参数, 这可以减少潜在的过度拟合.

为了进行跨图匹配操作, 还需要得到图的整体嵌入表示. 这里不是孤立地在一个图中聚合自身的全局上下文信息, 而是计算两个图节点细粒度匹配之后得到的整体嵌入表示. 以得到代码图的整体嵌入表示为例, 具体操作的表述如下.

首先, 计算文本图 G_q 的任一节点 q_i 与代码图 G_p 中的所有节点的跨图注意力分数.

$$\alpha_{i,j}=\cos(q_i,p_j),\forall j=1,\ldots,N \tag{6}$$

其中, $\alpha_{i,j}$ 作为代码图的节点 p_j 的权重, 计算代码图的所有节点嵌入的加权平均, 就得到了从文本图的某个节点视角, 代码图的整体嵌入表示的向量 $\tilde{\vec{h}}_p^i$, 也就是聚合了文本图节点信息和代码图自身信息的图嵌入. 此处节点之间的细粒度匹配操作传递了文本图节点的信息到代码图中, 丰富了代码图节点信息的表示.

$$\tilde{\vec{h}}_P^i=\frac{1}{N}\sum_j^N\alpha_{i,j}p_j,\ p_j\in\mathcal{V}_p \tag{7}$$

代码图的整体嵌入表示获得后即可应用多视角匹配函数 f_m, 对文本图的节点 q_i 和代码图的整体嵌入表示向量 $\tilde{\vec{h}}_p^i$ 进行对齐比较, 得到了文本图的节点 q_i 跨图匹配后的向量表示 $\hat{q}_i$.

$$\hat{q}_i=f_m\left(q_i,\tilde{\vec{h}}_P^i,w_m\right),q_i\in\mathcal{V}_q \tag{8}$$

相应地, 代码图的节点跨图匹配后的向量表示为 $\hat{p}_j$. 在对文本图和代码图的所有节点之间进行上述细粒度匹配操作后, 更新后的文本图节点嵌入和代码图节点嵌入分别为 $H_q=\{\hat{q}_i\}_{i=1}^M\in\mathbb{R}^{(M,\tilde{d})}$ 和 $H_p=\{\hat{p}_j\}_{j=1}^N\in\mathbb{R}^{(N,\tilde{d})}$. 其中 $\tilde{d}$ 是更新后的节点维度, 也即 f_m 的视角参数. M 和 N 表示文本图和代码图的节点个数. 更新后的节点嵌入表示在一定程度上衡量了两张图中每对节点之间的对齐信息, 有利于之后的相似度得分计算.

2.4 相似度得分计算

对代码图和文本图进行代码搜索需要比较它们的相似度得分. 基于上面更新后的图节点嵌入, 可以得到代码图或文本图的最终整体嵌入表示. 本文没有采用平均池化操作, 而是采用另一种聚合方法 *FCMax*, 即全连接层变换后的最大池化操作. 因为最大池化操作可以容易地比较并找出文本图和代码图节点的相似性或不相似性, 而平均池化操作会扁平化地聚合节点信息, 可能会忽视重要的节点.

$$S_q=FCMax(H_q)=maxpooling\left(FC\left(\{\hat{q}_i\}_{i=1}^M\right)\right) \tag{9}$$

$$S_p=FCMax(H_p)=maxpooling\left(FC\left(\{\hat{p}_j\}_{j=1}^N\right)\right) \tag{10}$$

有了文本图的整体嵌入表示 S_q 和代码图的整体嵌入表示 S_p, 很容易计算它们的相似度分数. 这里我们使用

余弦函数衡量相似度得分: $sim_1 = \cos(S_p, S_q)$. 所得 sim_1 即为应用基于多视角匹配函数的节点层面匹配策略后的相似度得分.

2.5 基于神经张量网络的图层面匹配策略

GraphSearchNet 的第 2 个不足是它虽然使用多头注意力机制获得代码和文本的全局信息, 但仍然没有进一步对代码和文本的全局信息进行比较和匹配. 为了应对第 2 个不足, GraphCS 提出了基于神经张量网络的图层面匹配策略, 将文本图和代码图送入神经张量网络衡量图对的相似性, 对代码图和文本图在多个维度上进行交互匹配, 能够有效捕捉到代码和文本的全局关系.

2.5.1 图全局上下文感知的嵌入表示

根据图神经网络的传播模型, 每个节点聚合的信息来自于自身和邻居的节点信息, 所以图中每个节点的信息是局部的. 在池化操作后获得的整体嵌入表示, 能很好地表示整张图所有节点的信息汇总和全局结构信息. 但是在图层面匹配策略中, 并没有采取池化操作获取图的整体嵌入表示. 本文提出了一种更有效的基于注意力的聚合方法. 并且图层面匹配策略使用的是文本图和代码图的初始节点嵌入 $H_q = \{q_i\}_{i=1}^{M} \in \mathbb{R}^{(M,d)}$ 和 $H_p = \{p_j\}_{j=1}^{N} \in \mathbb{R}^{(N,d)}$. 节点层面匹配策略得到的更新后节点嵌入含有两张图交互的信息, 这里应该使用它们的初始节点嵌入表示, 不受节点匹配层的干扰.

以代码图为例, 初始节点嵌入为 $H_p = \{p_j\}_{j=1}^{N} \in \mathbb{R}^{(N,d)}$. 首先计算节点嵌入简单平均的非线性变换: $C = \tanh\left(\left(\frac{1}{N}\sum_{j=1}^{N} p_j\right)W\right)$, 其中 $W \in \mathbb{R}^{d\times d}$ 是可学习的权重矩阵. 得到的 $C \in \mathbb{R}^d$ 表征了图的全局上下文信息. 基于 C 可以为每个节点计算一个注意力权重 a_j. 先取节点嵌入 p_j 与全局上下文向量 C 的内积, *Sigmoid* 函数的作用在于确保权重值在 (0, 1) 范围内. 很容易想到, 与全局上下文向量更相似的节点嵌入获得了更大值的权重, 这样得到的图整体嵌入表示更有意义. 得到每个节点的关注权重 a_j 后, 就可以通过公式 (11) 和公式 (12) 得到图的整体嵌入表示 $h_p \in \mathbb{R}^d$.

$$a_j = Sigmoid\left(p_j^{\mathrm{T}} \cdot C\right) \tag{11}$$

$$h_p = \sum_{N}^{j=1} (a_j \cdot p_j) \tag{12}$$

对代码图和文本图分别应用上述相同的操作, 得到了它们全局上下文感知的整体嵌入表示 h_p 和 h_q.

2.5.2 相似度得分计算

在图层面匹配策略中, 我们应用神经张量网络 (*NTN*)[21]探索文本图和代码图之间的全局关系. 与传统的线性网络相比, 神经张量网络 (*NTN*) 的优势是它能够在多个维度上比较两个嵌入向量的相似关系. 给定两个向量 h_i 和 h_j, *NTN* 使用双线性张量计算两个嵌入之间的多维度关系.

$$NTN_{(h_i,h_j)} = \sigma\left(h_i^{\mathrm{T}} \cdot W^{[1:k]} \cdot h_j + V\begin{bmatrix} h_i \\ h_j \end{bmatrix} + b\right) \tag{13}$$

其中, σ 是一个非线性激活函数, $W^{[1:k]} \in \mathbb{R}^{d\times d\times k}$ 是一个有 k 个切片的张量, $V \in \mathbb{R}^{k\times 2d}$ 和 b 是标准神经网络的权重矩阵和偏置. 向量 $S \in \mathbb{R}^k$ 是通过双线性张量积 $h_i^{\mathrm{T}} \cdot W^{[1:k]} \cdot h_j$ 计算得到的, 它的每个切片学习输入嵌入之间的不同相似性模式. 将代码图和文本图全局上下文感知的整体嵌入表示 h_p 和 h_q 输入一层神经张量网络, 输出了代码图和文本图全局互动的相似向量表示 $NTN_{(h_p,h_q)} \in \mathbb{R}^k$. 这个向量在 k 个维度上衡量了两张图全局视角下的相似性关系.

要得到代码图和文本图的最终相似度得分 sim_2, 还需要将 $NTN_{(h_p,h_q)} \in \mathbb{R}^k$ 送入多层全连接神经网络以及相应的激活函数层. 全连接神经网络使用多层加权神经元乘法将维度 m 的输入映射到所需的输出维度 n. 输入神经张量 $NTN_{(h_p,h_q)} \in \mathbb{R}^k$, 从多层全连接神经网络的最后一层, 输出了分数值标量 $sim_2 \in \mathbb{R}$. 这样就得到了衡量代码图和文本图全局关系的相似性分数 $sim_2 \in \mathbb{R}$.

2.6 模型训练

综合节点层面匹配策略和图层面匹配策略的得分结果, 应用如下的加权公式得到一对代码图和文本图最终的

相似度分数 $score(q,p)\in\mathbb{R}$. 其中 a_i 是一个权重参数.

$$score(q,p)=F(sim_1,sim_2)=\sum_{i=1}^{2}a_i\cdot sim_i \tag{14}$$

在搜索过程中, 代码库的所有代码片段图嵌入已经离线完成. 对于一个文本图 G_q, 可以与代码库中的所有候选代码图 G_p 并行计算, 并快速得到每对文本和代码最终的相似度得分 $score(q,p)$. 相似度得分高的候选代码片段会被返回, 整个搜索过程是快速高效的. 模型在大型的文本代码配对的语料库上以端到端的方式进行训练的. 具体来说, 训练语料库 $\mathbb{T}$ 中的每一个训练样本都是一个三元组 $<q,p,p^->$. 对于每个文本查询 q 和对应的代码片段 p, 随机从语料库中选中一个负样本代码片段 p^-. 模型的目标是预测得到 $score(q,p)$ 的分数比 $score(q,p^-)$ 的分数高. 本文使用 margin 损失函数进行模型优化.

$$\mathcal{L}(\theta)=\sum_{<q,p,p^->\in\mathbb{T}}\max(0,\delta-score(q,p)+score(q,p^-)) \tag{15}$$

其中, θ 代表要训练的所有模型参数, δ 代表损失函数的边际值. 排名损失会促使文本查询和相关的正样本代码片段的相似度得分上升, 而文本查询和负样本代码片段的相似度得分下降.

3 实验部分

3.1 数据集处理

本文在两个公开的代码搜索数据集上评估提出的 GraphCS 模型: FB-Java 数据集和 CSN-Python 数据集. FB-Java 数据集是 Fackbook 团队[22]发布的可用于代码搜索任务的 Android 项目代码数据集. GitHub 团队于 2019 年提出了评测代码搜索任务的基准数据集 CodeSearchNet Corpus[23], 这个庞大的数据集包含 6 种编程语言的海量代码文本对, 本文使用其中的 Python 数据集模块 CSN-Python. 要说明的是, 我们的数据集处理流程和图构建模块沿用 Ling 等人[24]的方法和数据.

为了使数据集更好地评估代码搜索任务, 需要对数据集的每个函数或方法做一些必要的处理和移除工作. 首先剔除没有文档描述和注释信息的代码片段, 这样的代码片段不适合用来训练模型. 代码行数少于 3 行或注释信息少于 3 个单词的代码片段也移除, 不便于图的构建. 对于有方法重载或覆盖的片段和有重复信息的代码片段, 只保留一种代码方法.

对于文本图构建模块, 可以应用 Stanford CoreNLP 工具包[25]生成注释文本的选区树, 并将选区树中的终端节点链接起来. 对于代码图构建模块, 基于文献 [26,27] 的开源代码构建增强表示的抽象语法树. 并且在图构建阶段, 有必要限制文本图和代码图的节点最大数为 300, 过于庞大的图对实验设备要求更高, 所以舍弃节点数大于 300 的文本代码对. 最后从 FB-Java 数据集中, 得到了 226 259 对 Java 代码图和文本图, 从 CSN-Python 数据集中, 得到了 330 404 对 Python 代码图和文本图. 然后, 将数据集分为训练集/验证集/测试集, 统计结果如表 2 所示.

表 2 数据集划分

数据集	训练集	验证集	测试集
FB-Java	216 259	9 000	1 000
CSN-Python	312 189	17 215	1 000

3.2 模型参数设置

在图嵌入模块设置关系图卷积网络的层数是 1, 输入的节点维度为 300, 输出的节点维度为 100. 值得注意的是, 本文的模型是基于 siamese 网络[28]进行训练的, 也就是说用于学习文本图和代码图表示的 RGCN 网络共享参数. 这样不仅可以减少模型过拟合的可能, 还能使节点规模大小不同但结构语义相似的文本图和代码图经过关系图卷积网络后更加接近.

文本图和代码图中每个节点都包含一个单词, 采用词嵌入工具 GloVe[29]得到初始化的单词嵌入向量, 每个单

词嵌入的维度为 300. 对于无法从 GloVe 初始化的单词, 如代码图中的 CamelCase, 一般先分割成子标记 Camel 和 Case, 再用 GloVe 预训练的子标记嵌入的平均值进行初始化. 在节点层面的匹配模块中, 多视角匹配函数的参数 $\tilde{d}$ 设置为 150. 在图层面的匹配模块中, NTN 中双线性张量网络的切片参数 k 设置为 50.

模型参数设置如下: 损失函数的 margin 值 δ 设为 0.5, batch size 设为 10, 并使用学习率 (learning-rate) 为 0.000 1 的 Adam 优化器[30]. 为了使模型有更好的泛化能力, dropout 参数值设为 0.1. 我们在深度学习框架 PyTorch 和图神经网络库 PyTorch_Geometric 上搭建模型, 并在配备有两颗 Nvidia RTX A5000 GPU 的服务器上训练模型. 这些参数都是在验证集上得出的最好结果, 关键的参数值如表 3 所示.

表 3 参数介绍

关键参数名称	参数数值
图节点词嵌入维度	300
RGCN输出节点维度	100
多视角匹配函数参数 $\tilde{d}$	150
神经张量网络切片参数k	50
batch size	10
margin	0.5
dropout	0.1

GraphCS 模型的时间复杂度主要取决于节点层面的匹配策略, 即计算公式 (7) 和公式 (8). 假设 M 和 N 分别代表文本图和代码图的节点个数, d 代表每个节点的维度, $\tilde{d}$ 代表多视角匹配函数的视角, GraphCS 整体的时间复杂度为 $\mathrm{O}(NMd+(N+M)\tilde{d}+(N+M)d\tilde{d})$.

3.3 评价指标

为了进行公平比较, 本文参考以往工作[22−24]的惯例, 将测试集的大小设置为 1 000, batch size 设置为 10, 每个批次中有 100 个候选代码片段. 对于测试数据集中的每一对代码片段和文本描述, 一般将文本描述看作真实查询, 并将对应的代码片段和其他 99 个随机选取的代码片段一起作为代码搜索任务的候选片段. 本文使用信息检索中常用的两种评价指标平均倒数排名 MRR 和在 k 处的成功率 $S@k$ 来衡量提出的模型和基线模型的性能. 具体来说, MRR 是一组查询结果 Q 的倒数排名的平均值. $FRank$ 指的是在给定长度的返回结果列表中首次出现符合要求的结果.

$$MRR=\frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{FRank_q} \tag{16}$$

其中, $FRank_q$ 指的是第 q 次查询的第 1 个命中结果的排名位置, $|Q|$ 是 Q 中的查询数量. 此外, $S@k$ 表示在排名靠前的 k 个排名结果中存在一个以上正确结果的百分比. 如果查询为真, Γ 值为 1, 否则值为 0.

$$S@k=\frac{1}{|Q|}\sum_{q=1}^{|Q|}(FRank_q\leqslant k) \tag{17}$$

对于这两个评价指标, 指标值越高说明代码搜索的结果越好, 也即模型的性能更好, 能够符合实际代码搜索场景的查询要求.

3.4 基线模型

为了评估提出的模型的有效性, 我们选择了 7 个基于深度神经网络的先进模型作为基线模型进行比较. 代表性的中文论文如文献 [7,12,14] 等, 代码未开源不便开展实验与之对比. 考虑的 7 个基线模型如下.

(1) NBoW (neural bag of words)、BiRNN、1D-CNN 和 self-attention 是文献 [23] 提供的 4 种基线模型. NBoW 使用词袋模型嵌入代码令牌, BiRNN 使用 GRU 作为代码序列的编码器, 1D-CNN 使用一维的卷积神经网络表征输入的代码令牌, self-attention 运用多头注意力机制计算代码序列中 token 的表示. 为了避免过度调优, 一般遵循作者的原始实验参数设置.

(2) Facebook 团队提出的 UNIF[31]通过词嵌入技术 fastText 嵌入代码和文本, UNIF 使用可学习的嵌入来映射代码和查询, 为了生成它们的表示, 代码片段的所有令牌嵌入都通过注意力加权聚合, 余弦距离用作相似度度量. 将其作为基线模型进行实验时, 设置词嵌入的维度为 100.

(3) DeepCS[8]从 3 个方面挖掘代码的语义信息: 方法名、代码中包含的 token、API 调用序列. 然后作者使用 LSTM 或 MLP 对 3 种不同的序列编码后融合, 对文本查询使用 RNN 嵌入. 由于他们对 API 调用序列的提取方法是基于 Java 语言特有的启发式方法, 对其他编程语言不起作用. 因此在实验中只使用方法名和代码标记作为源代码的语义特征.

(4) GraphSearchNet[15]用双向门控图神经网络 (BiGGNN) 训练了两个独立的程序编码器和查询编码器, 并通过多头注意力机制补充代码和文本的全局依赖信息. 作者将图的节点个数限制在 200 个, 而本文将图的节点个数限制在 300 个, 实验仍然遵循作者的原始设置.

3.5 与基线模型的对比

3.5.1 在测试集上的实验

为了证明我们提出的方法的有效性, 本节通过比较在 FB-Java 数据集和 CSN-Python 数据集上的平均倒数排名 MRR 和在 k 处的成功率 $S@k$ 来定量评估模型. 表 4 和表 5 给出了与 7 种基线方法在两个公开数据集上的实验数据对比. 黑体的数据表示最好的结果, 带有下划线的数据表示次好的结果. 最后两行 GraphCS (margin loss) 和 GraphCS (circle loss) 表示将损失函数分别设置为 margin loss 和 circle loss[32]的实验结果.

表 4 在 FB-Java 数据集上的实验结果 (%)

Model	*MRR*	*S*@1	*S*@5	*S*@10
NBoW	77.7	71.3	85.3	88.5
BiRNN	71.7	63.0	83.2	88.6
1D-CNN	22.6	12.3	32.7	45.7
Self-attention	65.3	54.4	79.1	84.2
UNIF	<u>84.8</u>	<u>78.1</u>	<u>92.5</u>	<u>95.7</u>
DeepCS	78.9	70.6	89.6	94.2
GraphSearchNet	71.3	64.5	88.6	89.6
GraphCS (margin loss)	**87.1**	**80.1**	**95.2**	**96.6**
GraphCS (circle loss)	**86.7**	**79.5**	**95.0**	**96.5**

表 5 在 CSN-Python 数据集上的实验结果 (%)

Model	*MRR*	*S*@1	*S*@5	*S*@10
NBoW	66.0	56.2	78.3	83.2
BiRNN	62.7	52.8	73.1	81.6
1D-CNN	18.4	10.5	25.1	33.6
Self-Attention	63.9	54.5	75.3	82.1
UNIF	70.1	59.7	83.8	<u>90.3</u>
DeepCS	64.4	52.2	78.2	88.3
GraphSearchNet	<u>73.9</u>	<u>65.3</u>	<u>84.2</u>	89.1
GraphCS (margin loss)	**88.3**	**80.6**	**97.7**	**98.8**
GraphCS (circle loss)	**87.7**	**80.1**	**97.2**	**98.6**

与所有的基线模型相比, 本文提出的 GraphCS 模型在两个数据集上的 4 个评价指标都展现了最好的性能. 可以看到, *MRR* 指标在 FB-Java 数据集和 CSN-Python 数据集上都超过了 85%, 这意味着 GraphCS 可以很容易地找到查询相关的代码片段. $S@k$ 是代码搜索任务的一个重要评价指标. $S@k$ 值越高, 表示在排名靠前的 k 个返回结果中存在正确结果的可能性越大. GraphCS 在两个数据集上达到了超过 95% 的 $S@5$ 数值, 即它有 95% 的概率从前 5 个返回的排名结果中得到正确的代码片段. 其他基线模型如 DeepCS 和 GraphSearchNet 在 $S@5$ 上都没有超过这个数值. 另外 GraphCS 模型在指标 $S@1$ 上也超过了 80%, 比其他模型至少高出了 10%, 即 GraphCS 模型找到的 Top1 结果有 80% 的概率是正确的代码片段, DeepCS 模型的 $S@1$ 数值是 52.2%, GraphSearchNet 模型的 $S@1$ 数值是 65.31%, 即第 1 个返回的代码片段有至少 35% 的概率不符合查询语句的要求, 或者返回的代码片段质量不高. 综上所述, 在精确查找时 GraphCS 的搜索精度更高, 返回的首个代码片段与查询的语义相关性更强. 同时, 我们也发现采用 margin loss 的结果要稍微好于基于 circle loss 的结果.

另外, 基线模型对于不同编程语言的搜索结果差异较大, 在 Python 数据集上的表现普遍低于在 Java 数据集上的表现, 而本文的模型表现稳定. 这可能是因为各个基线模型使用的嵌入编码方式对 Java 语言更适用, 语法简洁函数短小的 Python 语言的深层结构信息更不易挖掘.

3.5.2 真实代码搜索的样例展示与分析

为了更好地评估模型性能, 本节评价 GraphSearchNet 和 GraphCS 模型在真实的用户查询上的表现, 使用了

CodeSearch Challenge[23]的 99 个真实场景 Query 作为查询, 并剔除掉查询文本长度短于 3 个单词的 Query, 只选取前 50 个 Query 用于实验测试. 由于这时的查询来自于真实场景, 相应返回的代码片段也不应该来自于小型测试集, 应该是完全不同于训练语料代码库 CSN-Python 的另一个大型的代码库, 这里本文使用 CodeSearch Challenge[23]的代码搜索库, 其中包含大约 115 万个代码片段. 注意这里的代码搜索库量级是远远大于训练模型的代码库的, 并且二者包含的代码片段是没有交集的. 训练模型的代码库包含的是成对的代码片段和相关注释, 而这里的代码搜索库的代码片段是没有成对的注释的, 这样的数据集在面对真实查询时能够在一定程度上反应模型的泛化能力.

由于 FB-Java 数据集不是来自于 CodeSearch Corpus[23], 我们只在 CSN-Python 数据集上进行真实用户查询的评测. 为了直观地展示搜索效果, 以 *FRank* 作为评估标准. *FRank* 指标的结果符合人们浏览信息由上至下的准则, 其值越小就表示找到最好结果花费的搜索工作量越小, 也说明搜索结果越靠前. 因此 *FRank* 衡量单个查询搜索结果是有效的. 表 6 展示了两个算法搜索返回前 10 名结果的 *FRank* 值, NF 表示没有搜索到一个相关的正确结果.

表 6 在 *FRank* 上的评估结果

Query	GraphSearchNet	GraphCS
Convert int to string	1	1
String to date	3	5
Sort string list	3	1
Save list to file	2	1
Set working directory	2	3
Group by count	5	3
Socket recv timeout	3	NF
Convert decimal to hex	7	4
Export to Excel	7	NF
Convert JSON to CSV	3	2
Pretty print JSON	1	1
Replace in file	4	3
k means clustering	NF	4
Connect to SQL	2	1
HTML encode string	NF	2
Finding time elapsed using a timer	2	6
Parse binary file to custom class	NF	3
Get current IP address	5	4
Convert int to bool	2	5
Read text file line by line	1	1
Get executable path	4	7
httpclient post JSON	1	1
Get inner HTML	1	1
Convert string to number	1	1
Map to JSON	1	2
Parse JSON file	2	1
Get current observable value	7	4
Get name of enumerated value	NF	NF
How to empty array	2	1
How to get current date	1	1
How to make the checkbox checked	3	3
How to reverse a string	NF	3
Read properties file	NF	9
Copy to clipboard	1	1
Convert HTML to pdf	1	2
JSON to XML conversion	10	5
How to randomly pick a number	1	1

表 6 在 *FRank* 上的评估结果 (续)

Query	GraphSearchNet	GraphCS
Nelder mead optimize	NF	NF
Hash set for counting distinct elements	NF	NF
How to get database table name	1	3
Find int in string	3	2
Get current process id	2	4
Regex case insensitive	NF	5
Custom HTTP error response	5	3
How to determine a string is a valid word	1	3
HTML entities replace	2	NF
Set file attrib hidden	NF	4
Sorting multiple arrays based on another	NF	NF
String similarity levenshtein	8	6
How to get HTML of website	1	4

从表 6 中可以看出, GraphSearchNet 未搜索到结果个数为 11 个, GraphCS 未搜索到结果个数为 7 个, 并且 GraphCS 方法返回的在 Top1 处的符合查询要求的代码片段个数也更多. 从整体上分析, GraphCS 方法大多数搜索结果的 *FRank* 值更小, 相关的正确结果更靠前. 例如对于真实的查询语句 convert decimal to hex, GraphSearchNet 的 *FRank* 值为 7, GraphCS 的 *FRank* 值为 4, 意味着 GraphCS 的返回结果列表第 4 个符合查询要求, 而 GraphSearchNet 在结果列表第 7 个才找到符合要求的结果. 综上所述, 在真实场景 Query 测试集上的 *FRank* 值表明, 本文的方法相比 GraphSearchNet 有一定的进步, 可以搜索到更符合查询要求的代码片段.

3.6 语义匹配操作的影响

为了评估本文模型的语义匹配策略的有效性, 进行了消融研究评估以下模型变体的性能. RGCN 是指不采用任何语义匹配策略的模型变体, 在图编码之后对节点嵌入直接应用池化操作 FCMax, 并用余弦距离度量相似度分数. Node-level 是指单独应用节点层面的匹配策略, 对嵌入后的节点集基于多视角匹配函数进行匹配更新. Graph-level 是指单独应用图层面的匹配策略, 在得到图全局上下文感知的整体嵌入表示之后, 应用神经张量网络得到一对图的相似性向量并输入多层全连接神经网络得到相似度得分. 表 7 展示了它们的对比实验数据.

表 7 语义匹配操作的影响 (%)

Model	FB-Java dataset				CSN-Python dataset			
	MRR	*S*@1	*S*@5	*S*@10	*MRR*	*S*@1	*S*@5	*S*@10
RGCN	75.5	64.3	89.6	93.8	79.2	70.7	90.5	94.8
Node-level	84.4	79.1	93.1	94.9	85.6	79.8	94.5	96.2
Graph-level	71.2	70.5	81.9	85.1	72.9	71.8	86.8	89.3
GraphCS	87.1	80.1	95.2	96.6	88.3	80.6	97.7	98.8

从表 7 中可以看到, 采用匹配策略的模型 GraphCS 比模型变体 RGCN 取得了更好的搜索性能. 还可以明显看出, 节点层面的跨图匹配策略比图层面的全局匹配策略得分更高, 证明使用多视角匹配函数更新节点嵌入的策略在 GraphCS 模型中贡献的性能更多. 我们猜测, 神经张量网络虽然能够在多个维度上捕获图对的相似性模式, 但相似性向量通过多层全连接网络和激活函数得到的得分可能不如使用余弦函数获得的得分高. 另外, 直接使用图卷积神经网络编码而不应用任何匹配操作的简单模型 RGCN 还是比大多数基线模型性能更强大, 这证明了使用图结构数据和关系图卷积网络对于查询文本和代码片段都是有效的, 因为图学习能够捕获更多的结构语义信息.

3.7 候选代码片段的大小对模型效果的影响

本文参考以往工作[22−24]的惯例, 将候选代码片段设置为 100, 即一个正确的片段和 99 个随机选取的其他片段. 为了更好地验证候选代码片段的大小 *C* 对实验结果的影响, 本文进一步探索不同候选代码片段的大小 (即 *C*=100,

200, 300, 400, 500) vs. 实验结果的影响, 评价结果采用 *MRR* 和 *S@K*, 实验结果如图 6 所示.

从图 6 中可以看出, 当候选代码片段的大小设置为 100 时, 本文提出的方法在 FB-Java 和 CSN-Python 上均表现出最好的结果. 当候选代码片段的大小设置为 200, 300, 400 或 500 时, 所有评价指标 *MRR* 和 *S@K* 并没有明显的变化, 这表明本文提出的 GraphCS 具有良好的健壮性. 特别值得说明的是, *S@5* 和 *S@10* 的值始终保持在 95% 左右, 表明本文提出的 GraphCS 方法在面对实际场景的搜索时, 依然能够表现出良好的性能.

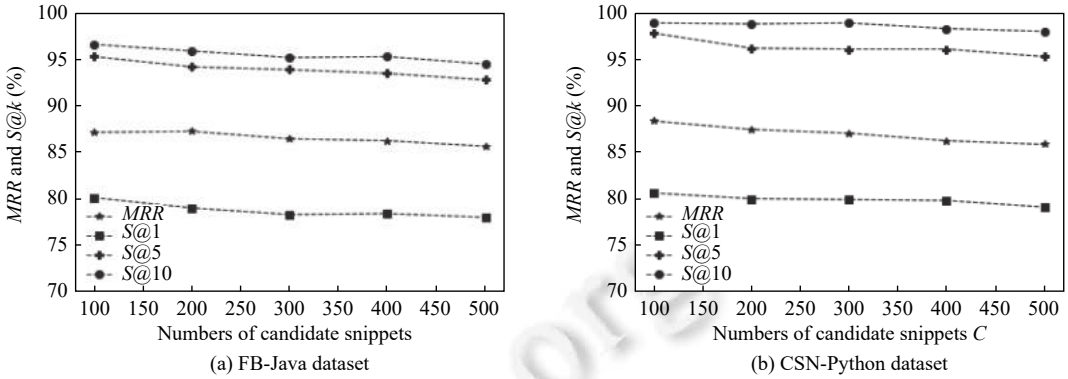

图 6 候选代码片段大小 *C* vs. 实验结果的影响

4 相关工作

近年来, 深度学习技术在许多不同领域取得巨大成功, 软件工程领域的研究者们应用自然语言处理方法提高了源代码相关任务的性能. 源代码相关任务与深度学习紧密相连的活跃研究领域包括代码摘要[33−36]、代码生成[37−39]和代码搜索[8−15]等, 这些任务的发展相互促进且都是有关联的, 预训练模型 CodeBERT[40]支持自然语言代码搜索和代码文档生成等多种下游任务, Chen 等人[10]提出的 BVAE 设计两个变分自动编码器用于生成更适合代码搜索的代码摘要. 本文主要关注对代码复用至关重要的代码搜索任务. 根据相关文献的发表时间线, 代码搜索大致经历了 3 个阶段: 2000 年以前基于软件工程规范和字符匹配的发展阶段, 2000–2016 年间基于信息检索技术的增长阶段以及 2016 年以后基于经典机器学习方法和深度学习技术的阶段. 本节主要介绍 2016 年以后出现的代码搜索技术.

ROSF[41]对代码片段基于信息检索方法 BM25 先进行粗粒度排序, 多项式逻辑回归模式用于细粒度的重新排序. Source Forager[42]挖掘代码片段的所有可能特征类, 并采用二元分类支持向量机的监督学习技术来计算不同特征类的相对重要性. SCOR[43]利用 Word2Vec 技术和马尔可夫随机域的组合考虑代码片段语义和标记的顺序. CodeMF[44]是一种特征融合的方法, 它通过核主成分分析等技术挖掘 StackOverFlow 的帖子来搜索高质量的软件库. 经典机器学习方法的应用从多个方面提升了代码搜索引擎的可信度和准确度, 但对文本和代码的理解还不够深入. DeepCS[8]是第 1 个将深度神经网络应用于代码搜索任务的工作, 该方法利用 MLP 或 RNN 学习查询文本和代码片段的表示, 解决了早期匹配技术对自然语言和代码片段语义理解的困难. 后续工作[9−11,23,31]和 DeepCS 较类似, 即先用不同的序列编码器将代码片段和查询文本映射为同一嵌入空间的向量, 然后计算它们的余弦相似度.

基于图嵌入的方法更进一步学习了代码片段的结构语义, 弥补了序列编码模型的不足. Gu 等人[45]后续还提出了一种基于图核的方法 CodeKernal 以选择合适的 API 使用范例, 它将源代码表示为对象使用图, 通过图核的方法将图聚类到连续的空间. Zeng 等人[46]提出的 DeGraphCS 模型将源代码的数据流控制流整合到基于变量的图中, 并通过改进的门控图神经网络得到图节点的嵌入. 本文与这些工作的不同之处在于: 1) GraphCS 将代码和文本都转化为图, 图结构在经过图神经网络学习之后可以通过全局视角进一步探索代码片段和文本查询的交互关系. 2) GraphCS 不仅可以捕获单个代码片段或查询文本的语义信息, 还能探索它们之间细粒度的语义匹配关系, 这有利于实际搜索场景的精确查询.

5 总结与展望

代码搜索是实现代码复用的关键, 也是智能化软件开发各项应用得以运行的基础[47]. 基于传统信息检索技术的代码搜索方法只能捕捉到文本和代码的浅层特征, 而运用深度神经网络得到代码和查询语言的向量表示, 然后对向量之间的空间距离进行优化能够挖掘更高层次的特征, 模型可以搜索出更加多样化的结果[48]. 本文提出的基于关系图卷积网络的代码搜索方法, 图表示能够完整保留代码片段的结构和语义信息, 提出的匹配操作能够探索文本图与代码图的细粒度匹配关系和全局关系, 相比基线模型在性能上有较大的进步.

候选代码库中可能会出现多个语义相同但形式不同的代码片段, 也会影响模型的性能, 这是一个值得关注但现有工作[8,15,22−24,44−46]尚未解决的难题. 因此, 在未来的工作中我们会重点考虑如何收集并标注高质量的代码搜索数据集, 探索语义相同但形式不同的代码片段对搜索结果的影响. 另外我们会考虑更多热门编程语言如 Javascript 数据集进行实验探究, 图编码模块可以采用针对代码结构有效的其他神经网络, 如基于代码数据流设计元路径, 使用 metapath2vec[49]算法对图结构编码, 进一步优化图节点嵌入模块. 我们还会重点关注代码搜索模型的可解释性问题, 以期对模型效果进行更合理的分析.

References:

[1] Liu BB, Dong W, Wang J. Survey on intelligent search and construction methods of program. Ruan Jian Xue Bao/Journal of Software, 2018, 29(8): 2177−2197 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5529.htm [doi: 10.13328/j.cnki.jos.005529]

[2] Bajracharya S, Ngo T, Linstead E, Dou YM, Rigor P, Baldi P, Lopes C. Sourcerer: A search engine for open source code supporting structure-based search. In: Proc. of the Companion to the 21st ACM SIGPLAN Symp. on Object-oriented Programming Systems, Languages, and Applications. Portland: ACM, 2006. 681–682. [doi: 10.1145/1176617.1176671]

[3] McMillan C, Grechanik M, Poshyvanyk D, Xie Q, Fu C. Portfolio: A search engine for finding functions and their usages. In: Proc. of the 33rd Int'l Conf. on Software Engineering. Honolulu: IEEE, 2011. 1043–1045. [doi: 10.1145/1985793.1985991]

[4] Lu ML, Sun XB, Wang SW, Lo D, Duan YC. Query expansion via WordNet for effective code search. In: Proc. of the 22nd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering. Montreal: IEEE, 2015. 545–549. [doi: 10.1109/SANER.2015.7081874]

[5] Fellbaum C. WordNet: An Electronic Lexical Database. Cambridge: MIT Press, 1998.

[6] Lv F, Zhang HY, Lou JG, Wang SW, Zhang DM, Zhao JJ. CodeHow: Effective code search based on API understanding and extended Boolean model (E). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 260–270. [doi: 10.1109/ASE.2015.42]

[7] Li X, Wang QX, Jin Z. Description reinforcement based code search. Ruan Jian Xue Bao/Journal of Software, 2017, 28(6): 1405–1417 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5226.htm [doi: 10.13328/j.cnki.jos.005226]

[8] Gu XD, Zhang HY, Kim S. Deep code search. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering. Gothenburg: IEEE, 2018. 933–944. [doi: 10.1145/3180155.3180167]

[9] Yao ZY, Peddamail JR, Sun H. CoaCor: Code annotation for code retrieval with reinforcement learning. In: Proc. of the 2019 World Wide Web Conf. San Francisco: ACM, 2019. 2203–2214. [doi: 10.1145/3308558.3313632]

[10] Chen QY, Zhou MH. A neural framework for retrieval and summarization of source code. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 826–831. [doi: 10.1145/3238147.3240471]

[11] Wan Y, Shu JD, Sui YL, Xu GD, Zhao Z, Wu J, Yu P. Multi-modal attention network learning for semantic source code retrieval. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 13–25. [doi: 10.1109/ASE.2019.00012]

[12] Ling CY, Zou YZ, Lin ZQ, Xie B, Zhao JF. Approach to searching software source code with graph embedding. Ruan Jian Xue Bao/Journal of Software, 2019, 30(5): 1481−1497 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5721.htm [doi: 10.13328/j.cnki.jos.005721]

[13] Tang J, Qu M, Wang MZ, Zhang M, Yan J, Mei QZ. LINE: Large-scale information network embedding. In: Proc. of the 24th Int'l Conf. on World Wide Web. Florence: ACM, 2015. 1067–1077. [doi: 10.1145/2736277.2741093]

[14] Huang SY, Zhao YH, Liang YM. Code search combining graph embedding and attention mechanism. Journal of Frontiers of Computer Science and Technology, 2022, 16(4): 844–854 (in Chinese with English abstract). [doi: 10.3778/j.issn.1673-9418.2010087]

[15] Liu SQ, Xie XF, Siow J, Ma L, Meng ZG, Liu Y. GraphSearchNet: Enhancing GNNs via capturing global dependencies for semantic code search. IEEE Trans. on Software Engineering, 2023, 49(4): 2839–2855. [doi: 10.1109/TSE.2022.3233901]

[16] Allamanis M, Brockschmidt M, Khademi M. Learning to represent programs with graphs. In: Proc. of the 6th Int'l Conf. on Learning Representations. Vancouver: OpenReview.net, 2018.

[17] Jurafsky D, Martin JH. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd ed., Upper Saddle River: Pearson Prentice Hall, 2009.

[18] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: Proc. of the 5th Int'l Conf. on Learning Representations. Toulon: OpenReview.net, 2017.

[19] Schlichtkrull M, Kipf TN, Bloem P, van den Berg, Titov I, Welling M. Modeling relational data with graph convolutional networks. In: Proc. of the 15th European Semantic Web Conf. Heraklion: Springer, 2018. 593–607. [doi: 10.1007/978-3-319-93417-4_38]

[20] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.

[21] Socher R, Chen DQ, Manning CD, Ng AY. Reasoning with neural tensor networks for knowledge base completion. In: Proc. of the 26th Int'l Conf. on Neural Information Processing Systems. Lake: Curran Associates Inc., 2013. 926–934.

[22] Li HY, Kim S, Chandra S. Neural code search evaluation dataset. arXiv:1908.09804, 2019.

[23] Husain H, Wu HH, Gazit T, Allamanis M, Brockschmidt M. CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv:1909.09436, 2019.

[24] Ling X, Wu LF, Wang SZ, Pan GN, Ma TF, Xu FL, Liu AX, Wu CM, Ji SL. Deep graph matching and searching for semantic code retrieval. ACM Trans. on Knowledge Discovery from Data, 2021, 15(5): 88. [doi: 10.1145/3447571]

[25] Manning C, Surdeanu M, Bauer J, Finkel J, Bethard S, MClosky D. The Stanford CoreNLP natural language processing toolkit. In: Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Baltimore: ACL, 2014. 55–60. [doi: 10.3115/v1/P14-5010]

[26] Fernandes P, Allamanis M, Brockschmidt M. Structured neural summarization. In: Proc. of the 7th Int'l Conf. on Learning Representations. New Orleans: OpenReview.net, 2019.

[27] Cvitkovic M, Singh B, Anandkumar A. Open vocabulary learning on source code with a graph-structured cache. In: Proc. of the 36th Int'l Conf. on Machine Learning. Long Beach: PMLR, 2019. 1475–1485.

[28] Bromley J, Bentz JW, Bottou L, Guyon I, Lecun Y, Moore C, Säckinger E, Shah R. Signature verification using a "Siamese" time delay neural network. Int'l Journal of Pattern Recognition and Artificial Intelligence, 1993, 7(4): 669–688. [doi: 10.1142/S0218001493000339]

[29] Pennington J, Socher R, Manning C. GloVe: Global vectors for word representation. In: Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing. Doha: ACL, 2014. 1532–1543. [doi: 10.3115/v1/D14-1162]

[30] Kingma DP, Ba LJ. Adam: A method for stochastic optimization. In: Proc. of the 3rd Int'l Conf. on Learning Representations. San Diego: ICLR, 2015. 7–9.

[31] Cambronero J, Li HY, Kim S, Sen K, Chandra S. When deep learning met code search. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 964–974. [doi: 10.1145/3338906.3340458]

[32] Sun YF, Cheng CM, Zhang YH, Zhang C, Zheng L, Wang ZD, Wei YC. Circle loss: A unified perspective of pair similarity optimization. In: Proc. of the 2020 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Seattle: IEEE, 2020. 6397–6406. [doi: 10.1109/CVPR42600.2020.00643]

[33] Li J, Li YM, Li G, Hu X, Xia X, Jin Z. EditSum: A retrieve-and-edit framework for source code summarization. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2021. 155–166. [doi: 10.1109/ASE51524.2021.9678724]

[34] Liu SQ, Chen Y, Xie XF, Siow JK, Liu Y. Retrieval-augmented generation for code summarization via hybrid GNN. In: Proc. of the 9th Int'l Conf. on Learning Representations. OpenReview.net, 2021.

[35] Zhang J, Wang X, Zhang HY, Sun HL, Liu XD. Retrieval-based neural source code summarization. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 1385–1397. [doi: 10.1145/3377811.3380383]

[36] Ahmad WU, Chakraborty S, Ray B, Chang KW. A transformer-based approach for source code summarization. In: Proc. of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, 2020. 4998–5007. [doi: 10.18653/v1/2020.acl-main.449]

[37] Brockschmidt M, Allamanis M, Gaunt AL, Polozov O. Generative code modeling with graphs. In: Proc. of the 7th Int'l Conf. on Learning Representations. New Orleans: OpenReview.net, 2019.

[38] Zhong RQ, Stern M, Klein D. Semantic scaffolds for pseudocode-to-code generation. In: Proc. of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, 2020. 2283–2295. [doi: 10.18653/v1/2020.acl-main.208]

[39] Sun ZY, Zhu QH, Xiong YF, Sun YC, Mou LL, Zhang L. TreeGen: A tree-based Transformer architecture for code generation. In: Proc. of the 34th AAAI Conf. on Artificial Intelligence. New York: AAAI Press, 2020. 8984–8991.

[40] Feng ZY, Guo D, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Findings of the Association for Computational Linguistics. ACL, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[41] Jiang H, Nie LM, Sun ZY, Ren ZL, Kong WQ, Zhang T, Luo XP. ROSF: Leveraging information retrieval and supervised learning for recommending code snippets. IEEE Trans. on Services Computing, 2019, 12(1): 34–46. [doi: 10.1109/TSC.2016.2592909]

[42] Kashyap V, Brown DB, Liblit B, Melski D, Reps T. Source forager: A search engine for similar source code. arXiv:1706.02769, 2017.

[43] Akbar S, Kak A. SCOR: Source code retrieval with semantics and order. In: Proc. of the 16th IEEE/ACM Int'l Conf. on Mining Software Repositories. Montreal: IEEE, 2019. 1–12. [doi: 10.1109/MSR.2019.00012]

[44] Hu G, Peng M, Zhang YH, Xie QQ, Gao W, Yuan MT. Unsupervised software repositories mining and its application to code search. Software: Practice and Experience, 2020, 50(3): 299–322. [doi: 10.1002/spe.2760]

[45] Gu XD, Zhang HY, Kim S. CodeKernel: A graph kernel based approach to the selection of API usage examples. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 590–601. [doi: 10.1109/ASE.2019.00061]

[46] Zeng C, Yu Y, Li SS, Xia X, Wang ZM, Geng MY, Bai LX, Dong W, Liao XK. $_{DE}G_{RAPH}$CS: Embedding variable-based flow graph for neural code search. ACM Trans. on Software Engineering and Methodology, 2023, 32(2): 1–27. [doi: 10.1145/3546066]

[47] Wang F, Liu JP, Liu B, Qian TY, Xiao YH, Peng ZY. Survey on construction of code knowledge graph and intelligent software development. Ruan Jian Xue Bao/Journal of Software, 2020, 31(1): 47–66 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5893.htm [doi: 10.13328/j.cnki.jos.005893]

[48] Hu X, Li G, Liu F, Jin Z. Program generation and code completion techniques based on deep learning: Literature review. Ruan Jian Xue Bao/Journal of Software, 2019, 30(5): 1206−1223 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5717.htm [doi: 10.13328/j.cnki.jos.005717]

[49] Dong YX, Chawla NV, Swami A. Metapath2vec: Scalable representation learning for heterogeneous networks. In: Proc. of the 23rd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Halifax: ACM, 2017. 135–144. [doi: 10.1145/3097983.3098036]

附中文参考文献:

[1] 刘斌斌, 董威, 王戟. 智能化的程序搜索与构造方法综述. 软件学报, 2018, 29(8): 2177–2197. http://www.jos.org.cn/1000-9825/5529.htm [doi: 10.13328/j.cnki.jos.005529]

[7] 黎宣, 王千祥, 金芝. 基于增强描述的代码搜索方法. 软件学报, 2017, 28(6): 1405–1417. http://www.jos.org.cn/1000-9825/5226.htm [doi: 10.13328/j.cnki.jos.005226]

[12] 凌春阳, 邹艳珍, 林泽琦, 谢冰, 赵俊峰. 基于图嵌入的软件项目源代码检索方法. 软件学报, 2019, 30(5): 1481–1497. http://www.jos.org.cn/1000-9825/5721.htm [doi: 10.13328/j.cnki.jos.005721]

[14] 黄思远, 赵宇海, 梁棘铭. 融合图嵌入和注意力机制的代码搜索. 计算机科学与探索, 2022, 16(4): 844–854. [doi: 10.3778/j.issn.1673-9418.2010087] [doi: 10.3778/j.issn.1673-9418.2010087]

[47] 王飞, 刘井平, 刘斌, 钱铁云, 肖仰华, 彭智勇. 代码知识图谱构建及智能化软件开发方法研究. 软件学报, 2020, 31(1): 47–66. http://www.jos.org.cn/1000-9825/5893.htm [doi: 10.13328/j.cnki.jos.005893]

[48] 胡星, 李戈, 刘芳, 金芝. 基于深度学习的程序生成与补全技术研究进展. 软件学报, 2019, 30(5): 1206–1223. http://www.jos.org.cn/1000-9825/5717.htm [doi: 10.13328/j.cnki.jos.005717]

周光有(1983－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为自然语言处理, 信息检索.

余啸(1994－), 男, 博士, 讲师, 主要研究领域为软件工程, 深度学习.

谢琦(1997－), 女, 硕士, 主要研究领域为自然语言处理, 软件工程.