计算机科学与探索
Journal of Frontiers of Computer Science and Technology
1673-9418/2022/16(09)-2011-19
doi: 10.3778/j.issn.1673-9418.2110073

基于深度学习的代码表征及其应用综述

张祥平[1,2],刘建勋[1,2+]
1. 湖南科技大学 服务计算与软件服务新技术湖南省重点实验室,湖南 湘潭 411201
2. 湖南科技大学 计算机科学与工程学院,湖南 湘潭 411201
+ 通信作者 E-mail: ljx529@gmail.com

摘　要:对程序进行分析、推理能够对软件开发、维护、迁移起到重要作用。如何高效地从程序代码中获取高质量信息成为了当前研究的热点。近几年有许多学者将基于深度学习的表征技术引入到程序代码分析任务中。深度学习模型能够自动地提取代码中所包含的隐含特征,降低对人工制定特征的依赖。首先介绍了代码表征的背景知识和基本概念,从代码静态信息分析角度出发,总结了基于深度学习的代码表征研究工作。之后进一步介绍了代码表征在代码克隆检测、代码搜索和代码补全三个任务上的具体应用。最后分析现有基于深度学习的代码表征工作中仍然存在的问题,并展望了未来可能的研究方向。

关键词:代码表征;表征学习;软件工程;代码分析;深度学习
文献标志码:A　　**中图分类号:**TP391

Overview of Deep Learning-Based Code Representation and Its Applications

ZHANG Xiangping[1,2], LIU Jianxun[1,2+]
1. Hunan Key Lab for Services Computing and Novel Software Technology, Hunan University of Science and Technology, Xiangtan, Hunan 411201, China
2. School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, Hunan 411201, China

Abstract: The analysis and inference of program play an important role in software development, maintenance and migration. How to efficiently obtain high quality information from program code has become a hot research topic. In recent years, a large number of researchers have introduced the deep learning-based representation technology into the code analysis tasks. The deep learning model can automatically extract the implicit and useful features implicit in the source code, which can alleviate the dependence on the manual construct feature. This paper first introduces the background and basic concepts of code representation, and summarizes the recent research work on deep learning-based code representation learning from the perspective of code static information analysis. Furthermore, this paper introduces the application of code representation on three tasks, code clone detection, code search and code completion. Finally, it discusses the challenges of deep learning-based code representation and the possible research directions in this field.

Key words: code representation; representation learning; software engineering; code analysis; deep learning

基金项目:国家自然科学基金(61872139)。
This work was supported by the National Natural Science Foundation of China (61872139).
收稿日期:2021-10-28　　**修回日期:**2022-04-21

计算机软件是现代社会最具代表性的产物，它被应用在人们生活工作中的方方面面。在现代计算机软件发展过程中，数以百万计的开发者参与并贡献了大量高水平的软件代码。然而由于代码本身所具有的抽象性、复杂性与可变性，想要进行高效、快速的软件开发仍是一个艰巨任务。针对此问题，开发者们研发出许多软件开发辅助工具，这些工具旨在帮助开发者理解、编写以及维护代码，从而提高开发人员的工作效率。

传统的软件开发辅助工具通常是基于人工制定的代码分析任务构建的，因此此类工具在特定任务范围内能够起到很好的效果。但随着代码开源活动的兴起，全球开源项目呈指数级增长。开源贡献者在开源平台贡献了大量高质量的开源项目，这些项目中包含了丰富的程序信息，如源代码、代码注释、故障报告以及测试用例等[1]。面对规模巨大、信息繁杂的代码数据，传统的代码分析方式难以获得令人满意的效果。Hindle等人[2]提出了代码的“自然性”假说，该假说认为程序语言与自然语言类似，包含了丰富的代码统计信息，这些信息可以用于构建功能更加丰富的软件开发辅助工具。代码“自然性”理论使人们意识到运用统计规律对代码信息进行分析是可行的，因此大量机器学习算法被运用到代码分析任务中，对代码进行多角度的信息挖掘，这极大地丰富了人们对于代码的认识[3-7]。在此过程中，最基础的环节是如何将代码转换为合适的代码表征向量，以适应不同的机器学习任务。

传统的代码表征方法简单地将代码视为自然语言进行处理，使用自然语言处理领域中成熟的算法将代码转换为表征向量，并在其上应用机器学习算法完成代码分析。如使用TF-IDF（term frequency - inverse document frequency）算法将代码转换为与词频相关的代码向量表征，此类方法仅仅使用简单的词频统计信息，没有将代码中丰富的语义信息以及结构信息融入到代码向量中，因此所生成的代码向量质量较低[8]。随着近几年深度学习取得的巨大成就，工业界与学术界对于在软件工程领域中探索和使用深度学习算法表现出了极大的热情。可针对不同的代码分析任务，使用不同的神经网络模型，生成高质量的代码表征向量[9]。如使用循环神经网络能够将变长的代码序列转换为固定长度的代码向量，进行代码分类和代码克隆检测任务[10]。使用序列到序列的神经网络模型进行代码自动生成任务，其中的编码器能够将代码编码为特征向量，之后使用解码器将其输出为代码序列[9]。

在海量代码数据上使用深度学习模型进行训练、学习、归纳，自动地提取代码中人工难以提取的深层次特征。通过不断组合不同的神经网络模型，将低层网络中获得的低阶特征组合成更加抽象的高阶特征，进而发现数据所隐含的高维信息。将这些信息融入到代码表征向量中，能够有效降低代码特征提取部分对人工制定特征的依赖[1,11]。与传统人工制定的代码特征提取方法不同，基于深度学习的代码特征提取方法无需依赖软件工程领域专家的背景知识，就能够从海量代码数据中自动地提取出适用于不同任务的隐含特征，进而避免了因为人类观察偏差所造成的失误[9,12-14]。

本文对近几年基于深度学习的代码表征工作进行分析与总结，并对代码表征工作下一步的研究方向提出了一些看法。

1 背景知识

如何从大量的数据中学习到数据之间的内在规律并应用于数据分析任务，是当前机器学习的首要目标。作为机器学习方法的基础，一个高质量的数据表征方法能够极大地提高机器学习算法的性能。机器学习算法在数据表征的基础上对模型进行优化，进而拟合已有的数据分布，最终应用于不同的任务场景[14-16]。现如今，由于计算机计算能力的提升，代码分析领域已从传统的人工制定特征提取方法转向对代码使用深度学习技术进行特征提取[10,17-19]。与传统的使用人工制定的特征提取方法不同，后者对代码分析的效率要远高于人工分析。首先，基于深度学习的代码分析方法可以减少人工制定特征的时间，降低了对代码分析的成本。其次，基于深度学习的代码分析方法能够发现更多人工难以发现的高维特征，也能够从不同的视角获取代码之间联系。

基于深度学习技术的代码分析方法通常可以用图1来表示其分析流程。图1包含四部分，分别是：(1)代码处理；(2)代码表征；(3)任务实现；(4)结果验证。

代码处理是从海量的代码库中收集代码，并根据任务的需要对代码进行简单的处理，如对代码进行漂亮打印（pretty-printing）[20]，将代码转换为抽象语法树（abstract syntax tree，AST）等[10,20]。代码表征是指将代码转换为其对应的向量。这个过程中主要是根据代码不同的粒度，将代码转换为对应的向量表

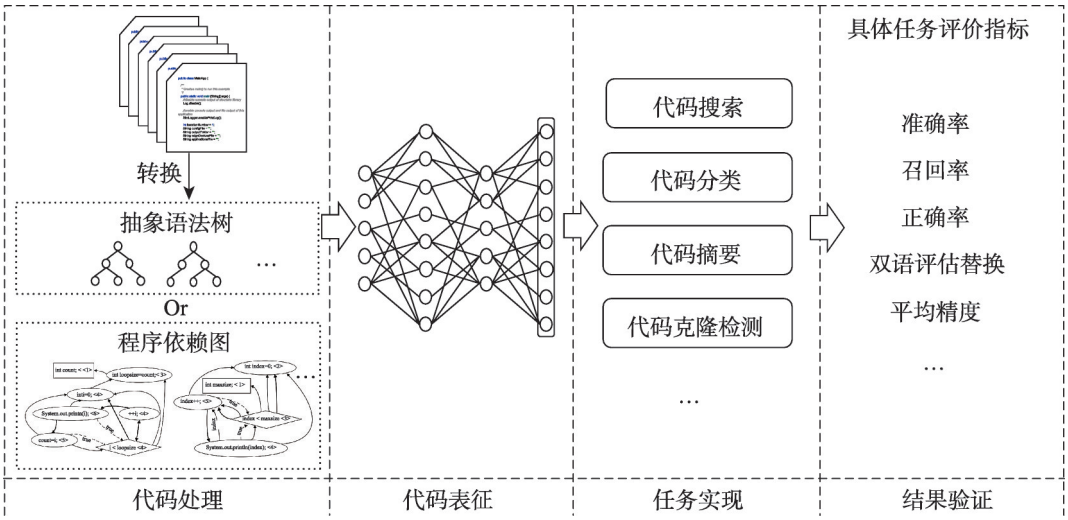

图1 基于深度学习的代码分析框架

Fig.1 Framework of code analysis with deep learning

示。这一步骤对之后机器学习模型的效果有极大的影响。目前最常见的方法是使用Word2vec算法将代码中词汇转换为对应的词向量，之后构建特定的神经网络模型对代码信息进行总结，得到代码的表征向量[19]。将神经网络用在所获得的代码表征向量之上，对具体的代码分析任务做预测分析，这一过程需要根据具体任务设计不同的网络结构。最后，对模型所生成的结果进行验证，通过不同的评价指标，对具体任务结果进行验证。

而在代码分析任务中，最重要的是对代码进行合适的表征，使其能够包含任务所需的有效信息。

1.1 基于矩阵的表征模型

基于矩阵的表征模型通常基于一个词共现矩阵 $\boldsymbol{F}$ 计算得到词的向量。词共现矩阵 $\boldsymbol{F}$，形状为 $W\times C$，其中 W 是词典大小，C 表示所使用的上下文信息长度。共现矩阵每一行 F_w 表示一个单词的分布表征，每一列 F_c 表示上下文信息。在文本处理中，词典大小 W 数量通常十分巨大。如图2，要在内存中维护一个规模庞大的共现矩阵将会消耗大量的内存资源。因此有许多工作研究如何构建合适的词共现矩阵以便减少计算资源的消耗。这类工作的目标可以用以下语言进行描述：使用一个映射函数 g，将词共现矩阵 $\boldsymbol{F}$ 映射为 f，即 $f=g(\boldsymbol{F})$，f 的形状为 $W\times d$，使得 $d\ll C$。

目前已有一些工作关注于对共现矩阵的降维。如Sahlgren[21]通过控制滑动窗口所滑动的方向、滑动窗口的大小来选取目标词不同范围的上下文信息，构造一个维度较小的共现矩阵 f，进而减少计算资源的使用。这个方法从共现矩阵的行（词汇个数）出发对共现矩阵进行降维。还有一些方法从共现矩阵的列（上下文信息）出发，如常见的分布表征方法，隐含语义分析模型（latent semantic analysis，LSA）[22]、

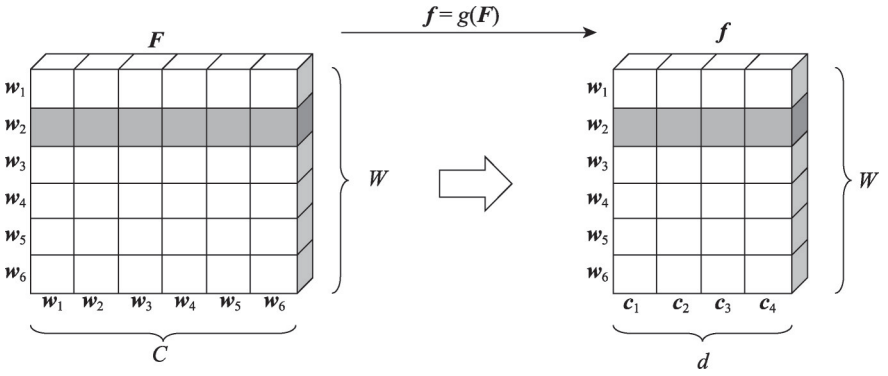

图2 基于矩阵的分布表征

Fig.2 Matrix-based distributional representation

隐含狄利克雷分布模型(latent Dirichlet allocation, LDA)[23]。比如LDA模型,在构建共现矩阵 ***F*** 的列时,所采用的是文档上下文信息而不是词信息,因而可以获得更高层次的语义信息。但是,将这些方法应用于代码表征任务中对所使用的代码语料库质量要求较高。如果所使用的代码库质量不高,如出现数据稀疏,包含较多噪声数据,使用上述方法难以获得高质量的代码表征结果。现有的一些方法研究如何在有限的内存资源中维护规模巨大的共现矩阵,如Sojka等人[24]提出一种增量构建方法,使得在超过2.7亿个词的语料库上也可以进行LSA和LDA主题建模。

1.2 基于神经网络的表征模型

基于神经网络的表征模型能够将不同类型的数据映射到连续、低维的向量空间中。由于其生成的向量是将原始数据的特征分布式表示在向量的各个维度上,其所生成的表征向量也可以称为是分布式表征。在自然语言处理领域中对文本的表征通常是集中在单词、句子和段落层面上[25]。文本的分布式表征极大地降低了表征向量维度,如使用独热表征(one-hot)需要 $O(N)$ 个参数才能区分 $O(N)$ 个输入空间(N 表示文本数据集中不同词汇的数量),而如果采用分布式表征,同样使用 $O(N)$ 个参数却能够区分 $O(2^N)$ 个输入空间[26]。并且分布式表征向量包含文本的语义信息,可以通过计算两个文本的表征向量之间的余弦距离来获得两个文本之间的相似度。最常见的分布式表征方法有Word2vec算法[27]。Word2vec通过固定大小的滑动窗口来获得中心词的局部上下文信息。Word2vec算法包含两种训练框架Skip-gram和CBOW,其对应的模型结构图如图3所示。Skip-gram框架根据所给中心词来预测其周围的词。

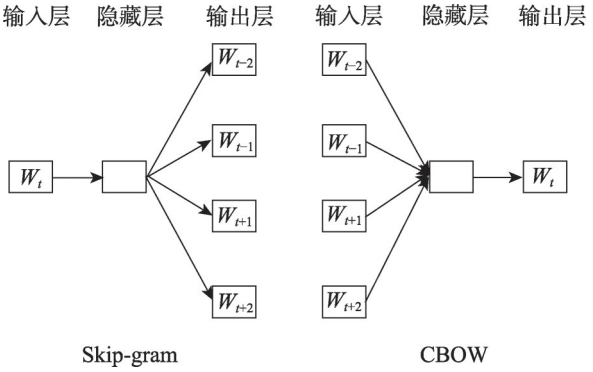

图3 Word2vec的两种训练框架

Fig.3 Two training frameworks of Word2vec algorithm

因此其优化的目标函数为:

$$L(\theta)=\prod_{j=1}^{S}\left(\prod_{i_j=1}^{T_j}\left(\prod_{-c_{ij}<u_{ij}<c_{ij},j\neq 0} p(w_{u_{ij}+i_j}|w_{i_j})\right)\right) \tag{1}$$

其中, $L(\theta)$ 表示损失函数 L 在参数 θ 下的值,其中 L 表示损失函数, θ 表示Word2vec训练模型中的所有参数, S 表示的是由 S 个句子组成的语料库, T_j 表示第 j 个句子所包含的单词个数, $w_{u_{ij}+i_j}$ 表示的是第 j 个句子中第 i 个单词 w_{i_j} 左右 c_{ij} 个单词之中的任意一个单词。

而CBOW框架是根据周围的词去预测中心词,因此CBOW框架的目标函数为:

$$L(\theta)=\prod_{j=1}^{S}\left(\prod_{i_j=1}^{T_j}\left(p(w_{i_j}|context_{i_j})\right)\right) \tag{2}$$

其中, S 表示的是由 S 个句子组成的语料库, T_j 表示第 j 个句子所包含的单词个数, w_{i_j} 表示的是句子 j 中第 i 个待预测词, $context_{i_j}$ 表示的是词 w_{i_j} 上下文信息。

2 代码的表征方法

现有的代码分析工作通常是从以下两个角度对代码进行分析[28]:(1)代码的静态分析;(2)代码的动态分析。其中代码的静态分析指基于代码的静态属性对代码进行特征提取,如代码的符号序列信息、API调用序列信息、代码对应的抽象语法树以及控制流图中存在的结构信息。而代码的动态分析关注于代码运行过程中产生的中间结果,基于中间结果对代码任务进行分析。本文所研究的内容是静态情况下代码信息的抽取与表征,因此本章内容将从不同的代码静态信息出发,分析深度学习技术在不同代码分析任务中所起到的作用。而对于代码的动态分析方法,本文不作详述。

2.1 基于符号序列的代码表征

在基于符号序列的代码表征方法中,最基础的步骤是根据语法规则将代码中的词汇单元(token)划分出来。在这个过程中需要用到词法分析器,词法分析器能够按照预定的语法规则将代码中的字符串分割为一个个词汇单元,这些词汇单元通常包含关键字、数字、标识符等。在此过程中,代码中的空白符也会被移除。将代码表示为词汇单元序列之后,利用深度学习技术对其进行建模,学习代码序列中所包含的有效信息,如功能语义信息、语法结构信息等,最后生成具有丰富代码信息的表征向量,应用于

不同的代码分析任务[1]。

Harer等人[29]在C/C++的词汇单元上使用Word2-vec算法用于软件漏洞的预测。其中词汇单元的表征向量用于初始化TextCNN模型进而用于代码分类。White等人[14]也使用Word2vec模型生成Java词汇单元的表征向量用于自动代码修复。Chen等人[30]使用Word2vec生成Java词汇单元表征向量用于在自动程序修复任务中找到正确的部分，该方法通过计算不同代码表征向量之间的余弦相似度来找到相似的代码片段。Henkel等人[31]从C语言项目中使用Word2vec生成抽象符号轨迹表征向量。他们在符号执行过程中收集轨迹信息，然后对轨迹中的符号名称进行重命名以降低词表大小。Nguyen等人[32]研究了Java和C#中API的表征向量，并用它发现了两种语言之间相似的API用法。他们使用Word2vec生成API元素嵌入，并且基于已知的API映射，该模型可以在两种语言中找到使用相似的簇类。将源代码翻译为API序列，并使用Word2vec从抽象的API序列中学习了API嵌入。Pradel等人[33]使用代码片段嵌入来识别潜在的错误代码。为了生成训练数据，作者从代码库中收集代码示例，并应用简单的代码转换来插入一个人工构造的错误，之后生成了正反码示例的嵌入，并用于分类器的训练过程。以上方法通过使用Word2vec算法，预先将代码中的词汇单元转换为实数向量，便于之后的深度学习模型进行特征提取。然而，受Word2vec算法的限制，所生成的词汇单元向量仅仅保留了基本的词共现信息，对于代码中存在的逻辑信息，如变量定义与使用的先后顺序，循环语句中的逻辑判断信息都无法获取。因此在代码表征的过程中，Word2vec算法通常只在最初的词汇单元转向量阶段使用，对于代码中包含的词法和语义信息，仍需要学习能力更强的神经网络模型才能进行完整的信息提取。

高质量软件项目中，通常每份代码都会有其对应的代码注释。而实际情况下，编写代码注释同样会消耗软件开发者大量的开发时间。因此目前也有很多学者研究如何自动地生成代码对应的注释以减少开发者的工作内容。目前有许多工作基于Seq2seq框架构建模型完成该任务。Seq2seq模型是一种能够根据给定的词汇序列，通过特定的方法生成另一种词汇序列的方法，比如在机器翻译中，使用Seq2seq模型可以将中文文本序列转换为其对应的英文文本序列。而在代码注释生成领域中，Seq2seq模型使用编码器（encoder）对代码中的词汇序列进行编码，得到上下文向量的中间表示，之后使用解码器（decoder）对上下文向量进行解码，从而生成这些代码序列所对应的功能注释。其中编码器与解码器通常是使用序列神经网络作为基本的序列特征提取模型。例如，Iyer等人[34]提出了CODE-NN模型，该模型使用附加注意力机制的LSTM序列神经网络模型对代码词汇序列进行编码。在编码过程中注意力机制会对代码中的每一个词汇单元都附加上注意力权重，最后获得代码对应的向量表征。如要进行代码注释生成工作，只需直接对上下文中间向量使用解码器即可生成对应的代码注释。该方法在C#和SQL所构成的代码数据集上进行代码注释生成，在METEOR和BLEU两个指标上取得了当时最优的结果。考虑到注意力机制对模型输入输出长度不同的数据有较好的建模效果，Allamanis等人[35]使用基于注意力机制CNN对代码进行建模，该模型将函数表示为词汇单元序列，在词汇单元序列上进行卷积操作，考虑到注意力机制能够获取代码序列中不同词汇的重要程度，在模型中使用了注意力机制。以上有关于代码注释生成的工作不同于传统的基于人工定制规则的方法，基于神经网络的代码生成模型能够挖掘更深层次的代码信息，获得更加丰富的代码表征向量用于下一步的代码分析任务。但是这些方法也存在明显的缺点，因为代码具有较强的逻辑结构关系，如不同API之间的调用关系、语句的执行顺序等。如果仅仅考虑代码中文本信息进行模型的训练，那么代码中的结构信息大部分都会丢失。

代码补全任务是根据已有代码词汇序列预测下一个词汇或者下一段代码序列。循环神经网络模型能够较好地对序列数据进行建模，因为其能够在任意长度的上下文窗口中存储、学习和表达相关信息，并且在时间序列上有延拓。现在很多工作使用循环神经网络对代码序列这类长文本数据进行特征提取。例如White等人[13]使用RNN（recurrent neural network）神经网络对代码中的词汇单元序列进行建模。作者在Github网站上收集了16 221个Java项目进行模型的训练。最终在Java代码的代码补全任务上取得了72.2%的准确率，这一结果远远高出了当时的N元语法模型所取得的效果。Li等人[36]考虑了在代码补全任务上超出词表问题（out of vocabulary, OoV），在传统的RNN神经网络的基础上增加了注意力机制，结合指针网络能够复制已出现过的上下文信息

的特性,实现了超出词表外代码片段的补全任务。在两个基准数据集上的实验证明该方法中注意力机制和指针混合网络在代码补全任务中的有效性。Bhoopchand等人[37]针对代码中不同标识符之间存在的长依赖问题,提出了基于指针网络的程序语言模型。该方法采用RNN的变种模型LSTM神经网络作为构建神经语言模型的基础模型。LSTM模型作为RNN模型的一个变种模型,能够在一定程度上缓解长文本中词汇之间的依赖问题(如,一个变量在代码开头就被定义,但是直到程序末尾时才被使用)。该方法通过使用指针网络在之前的输入数据中选择已出现过的标识符作为输出的备选结果,最后通过一个选择器综合考虑语言模型和指针网络两者的结果,共同预测输出词语的概率分布。该工作从Github网站上收集了4 100万行Python代码,在该数据集上进行代码补全实验,实验结果显示LSTM模型的补全预测结果能够提供距离当前补全位置较远的上文中所出现过的标识符,能够取得更高的补全准确率。与传统的基于统计规则的模型相比,这类基于词汇序列的神经网络模型的效果明显优于前者。但这些模型也存在许多不足,若是将代码简单地转换为词汇序列,那么代码中不同语句之间的顺序信息、不同API的调用信息等代码所特有的结构信息就会丢失,这势必会影响代码分析任务的性能。同时,受制于所使用基本模型本身的缺陷,如要使用神经网络对文本信息进行处理,必须要先确定最大文本长度以适应模型的输入,这同样会增加代码信息的提取难度。

代码搜索指根据用户对代码功能的描述语句在代码库中检索符合功能的代码。传统的代码搜索大多采用的是信息检索领域中的检索技术,通过计算查询语句与代码之间的相似程度进行搜索。最简单的方法就是关键词匹配。随着深度学习技术在各个领域所取得的突破性进展,也有大量学者将其应用于代码搜索领域。运用深度神经网络将代码与自然语言查询语句转换为同一个语义空间中的向量,通过对向量之间的相似度计算,能够有效地挖掘出代码与查询语句之间更高层的联系。深度学习在代码搜索上的应用通常是在代码的词汇序列上使用RNN神经网络模型进行建模,将词汇序列转换为序列向量。同时使用RNN神经网络模型对自然语言查询语句进行建模,将其转换为查询语句向量。在检索时,在同一向量空间中搜索与查询语句向量相似的代码向量作为检索结果。在此过程中,最重要的研究内容就是如何将序列表征向量与自然语言查询语句的向量映射到同一代码语义向量空间中。Gu等人[16]首先将深度学习技术应用在代码搜索领域,对给定的自然语言查询语句生成其所描述的API用法序列。该方法将问题转换为自然语言中的机器翻译问题,使用RNN神经网络对自然语言查询语句进行向量表征。同时使用不同模型对代码的不同粒度的信息进行表征建模,使用RNN模型对代码的词汇单元序列进行表征,使用RNN模型对代码的方法名进行表征,对代码中的API序列使用MLP模型进行表征。为了获得三个向量中最显著的特征,对这三个向量使用了最大池化操作,最后将这三个向量进行拼接,作为代码最终的向量表示。该工作使用了三元组损失函数(triplet loss),考虑了查询语句、与查询语句对应的代码(正例)、与查询语句不对应的代码(负例)三者之间的关系,通过最大化正例之间相似度以及最小化负例之间的相似度来训练模型,使得模型能够具备将不同源信息映射到同一语义向量空间中的能力。作者在Github网站上收集了1 820万条带有注释的Java方法代码片段进行实验。实验结果表明,该方法与其他传统的基于文本匹配的方法相比,搜索的准确度上取得了显著的提升。Shuai等人[38]考虑到自然语言查询与代码之间的语义关系,提出了一种共同注意力表征学习模型,用于同时学习代码与自然语言查询语句之间的关联表征。该方法构建了一个代码和查询序列的相关矩阵,通过按行或者按列的最大池化操作获得它们之间的语义关系。同样的,在文献[16]的数据集上进行实验,实验结果在平均排名倒数(mean reciprocal rank,MRR)指标上得到了26.72%的提升。

2.2 基于API调用序列的代码表征

基于应用程序接口(application programming interface,API)序列的代码表征方法是将代码中的API调用序列作为研究对象。当开发者对所需要使用的开发框架或库不熟悉时,需要学习具体框架的API使用方法,尤其是当使用规模庞大的框架如JDK(Java development kit),成千上万个API大大增加了软件开发难度。如果能够通过输入自然语言查询语句获得API的使用方法,将能极大地提高软件开发效率。现有的与API调用序列有关的工作将API序列作为研究对象。对程序中包含的API序列进行建模,获得API序列的表征向量,之后通过对比自然语言和代码API序列向量的相似度来完成API序列检索。

Gu等人[39]提出了基于深度学习的API检索方法，该方法根据自然语言查询语句来检索与其匹配的API序列。作者将API检索问题转换为机器翻译问题，通过使用Seq2seq框架构建一个自然语言到API序列的模型。首先将自然查询语句视为源语言，将自然语言查询语句中的每一个单词转换为词向量，之后使用RNN模型构建一个编码器。该编码器计算每个单词的隐藏状态，并且根据前一个单词的隐藏状态去预测下一个可能的输出单词，最后得到上下文向量 $\boldsymbol{C}$ 。同时，也使用RNN模型构建一个解码器，将编码器所生成的上下文信息 $\boldsymbol{C}$ 作为该解码器的初始隐藏状态，每个时间步都会生成一个单词作为API调用序列的内容。在模型训练中使用了束搜索（beam search）算法，以最小的代价来搜索最优的API序列。该方法在其处理好的751万对数据上进行实验，利用BLEU（bilingual evaluation understudy）来评估生成的API序列的质量，结果表明该方法较其他方法取得了40%左右的提升。Lu等人[40]提出了一种用于恶意软件行为分析的深度学习和机器学习组合模型。其中一部分分析了API调用序列之间的依赖关系，另一部分采用基于残差的双向LSTM神经网络获取API序列中的冗余信息。这种组合方式显著提高了恶意软件检测精度。

Saifullah[41]提出了一种使用带注意力机制双向LSTM的模型，将代码的词汇信息、句法信息以及上下文语义信息相融合，用于生成API调用序列。在整个过程中，作者首先对原始代码进行预处理，包括代码中方法名的抽取、变量名称规范化、参数规范化。在处理好的数据上使用基于注意力机制的双向LSTM编码器进行编码，生成上下文信息向量 $\boldsymbol{C}$ ，之后在该向量上使用解码器进行解码，同时采用了束搜索策略，生成最优的API调用序列。

在代码注释生成任务中，Hu等人[42]从代码中抽取出API序列以及摘要，使用Seq2seq模型对两者进行预训练。将API序列作为Seq2seq模型的输入，而代码摘要作为Seq2seq模型的训练目标，这一步骤使得生成的API序列表征与摘要表征映射到相近的向量空间。同时，该方法也采用了与CODE-NN模型类似的结构对代码词汇单元序列和摘要进行训练，将这些知识输入到网络模型中辅助代码注释的生成过程。该工作的创新点在于将预先构建的Seq2seq模型对API序列进行预训练，并且将该预训练模型中的编码器部分作为后面模型的编码器，通过将其拼接到代码token序列和摘要训练模型的解码器上生成代码摘要。该工作取得了当时最优的结果。

2.3 基于抽象语法树的代码表征

抽象语法树（AST）是源代码的抽象语法结构的树状表示，可以有效地表示程序的语法及其结构。抽象语法树并没有包含代码中所有的真实语法信息，它忽略了一些不重要的细节，只保留了必要的节点信息。树中的每个节点都对应源代码中的一种结构。在抽象语法树中，非终结符对应于树的中间节点（如Assign、If、For等），与代码片段的具体类型相对应，与程序结构紧密相关；而终结符则位于树的叶子节点（如字符串、变量名、方法名等），与程序语义密切相关[1]。抽象语法树可以被用于构建代码优化插件，如对代码进行语法检查，对代码进行格式化，对代码编写风格进行检查等。不同的编程语言有不同的抽象语法树构建工具，表1列出的是目前流行抽象语法树构建工具。

表1 不同编程语言的抽象语法树生成工具

Table 1 Tools of AST generation for different programming languages

编程语言	工具名称	工具地址
Java	Javaparser	http://Javaparser.org/
Python	astor	https://github.com/berkerpeksag/astor
TypeScript	TypeScript AST Viewer	https://ts-ast-viewer.com/
JavaScript	Javascript-astar	https://github.com/bgrins/Javascript-astar
C	pycparser	https://github.com/eliben/pycparser
C++	cppast	https://github.com/foonathan/cppast

利用深度神经网络对抽象语法树进行建模得到其向量表示，根据该特征向量完成代码模式检测任务。在克隆检测任务中，White等人[12]提出了基于循环神经网络的代码克隆检测方法，该方法将代码分为词汇以及句法两个层次。对于词汇级别的信息，作者在代码的词汇单元序列上使用RNN神经网络进行建模。而对于代码的句法级别的信息，作者首先将代码转换为其对应的抽象语法树结构，之后将抽象语法树转换为其对应的满二叉树，最后作者将满二叉树转换为Olive Trees，并在其上使用另一个RNN神经网络进行建模。该文将这两个特征相结合作为整个程序的特征向量，根据该向量进行代码克隆检测任务。Mou等人[19]提出了一种基于树的卷积神经网络模型（tree-based convolutional neural network，

TBCNN)。该模型考虑到不同代码对应的抽象语法树是不同的这一现象,采用了“连续二叉树”的概念,直接在代码所对应的抽象语法树上进行卷积操作。在卷积操作之后获得了不同数目的AST结构特征向量,由于数目不同不能直接作为神经网络的输入,因此该方法还采用“动态池化”技术,最终将数目不同的特征向量转换为了一个向量。TBCNN是一个通用的代码表征生成模型,所生成的向量能够包含代码片段中特有的代码模式,因而可以应用于不同的代码分析任务中。如文中使用所生成的代码表征向量用于代码功能分类任务,在POJ104[19,43]数据集上(由C语言编写的104个任务的代码数据集),分类任务的准确率能够达到94%。Wei等人[44]提出了CDLH(clone detection with learning to Hash)方法,该方法在抽象语法树的基础上使用LSTM模型,通过共享权重的方式学习两个代码片段之间的表征向量,这种模型使其能够检测出第四类代码克隆类型。Chen等人[45]同样采用了基于树的卷积神经网络,该方法考虑到已有的方法仅仅使用抽象语法树会造成代码语义信息的丢失,因此该方法将API的调用信息作为补充信息合并到抽象语法树中,之后进行代码表征。在POJ104和BCB[43]数据集上进行实验,在F1指标上获得了0.39和0.12的提升。Zhang等人[10]提出了一种基于抽象语法树的神经网络代码表征方法,该方法将代码转换为其对应的抽象语法树,不同于传统的基于抽象语法树的代码表征方法,该方法对抽象语法树进行语句级别上的切割,将完整的抽象语法树分割为多个语句树。针对每个语句树,该方法设计了语句编码器用于将语句树转换为对应的语句表征向量,通过使用双向GRU(gated recurrent unit)神经网络对语句向量进行建模,对双向GRU层输出的隐含状态向量进行最大池化(max-pooling)操作,以获得最显著的代码特征。其中,将抽象语法树分割为多个语句树变相地缓解了由于抽象语法树规模过大所带来的长依赖问题。该方法所生成的代码表征向量被应用于代码克隆检测任务中,在BCB数据集和POJ104数据集上取得了当时最好的检测结果。Wang等人[46]同样考虑了仅仅使用代码的抽象语法树进行代码表征建模实际上仍然有代码结构上的缺失这一问题,构建了代码抽象语法树的图形表示,通过将抽象语法树各个叶子结点相连构建出适合图神经网络处理的数据。上述方法均在抽象语法树上进行代码的表征学习,力图充分提取代码中的结构信息与语法信息。但是为了获得代码的结构信息,这些方法首先都是将代码转换为对应的抽象语法树,才能够将其作为深度学习模型的输入,同时由于深度学习模型本身的局限性,如循环神经网络仅能处理序列数据,还需要将树形数据转换为序列数据,而在这个过程中必定存在结构信息的丢失。

Hu等人[47]提出了基于AST的代码注释生成模型,该方法同样是将代码转换为其对应的抽象语法树,同时为了能够将抽象语法树作为神经网络模型的输入,作者还提出了一种基于抽象语法树的结构遍历方法,进而将抽象语法树转换为节点序列。并且作者也考虑了超出词表问题,当抽象语法树的节点类型和值不在预先定义的词表中时,使用节点的类型来替换该词。最后,使用基于注意力机制的Seq2seq框架完成注释生成任务。Alon等人[17]利用抽象语法树路径(AST path)来表示程序,通过AST路径来表示程序的结构和语法。该方法首先将代码转换为对应的抽象语法树,从抽象语法树中提取不同的路径信息,将路径上所有结点的值构成的元组称为路径上下文信息(path-context)。将每个路径上下文信息中的结点进行嵌入得到节点向量,之后将这些向量拼接成一个向量来代表这个路径上下文信息。同时作者还考虑使用注意力机制关注不同权重的路径信息。该方法在变量名预测、方法名预测以及表达式类型预测任务中都取得了目前最好的结果。之后Alon等人[48]又对该模型进行了改进,使用双向LSTM对抽象语法树的路径信息进行建模,将该模型应用于代码摘要任务中。Alon等人[49]还在代码补全任务上同样使用基于抽象语法树路径信息的模型,该模型通过估计抽象语法树上不同节点的条件概率来预测下一个节点的信息,进而完成代码补全任务。

2.4 基于图的代码表征

图的结构可以对程序中的数据流动关系进行建模,例如控制流图以及数据依赖图。图中节点表示程序中的元素,例如程序中的token或变量、程序中指针(pointer)的内存地址,图中的边表示节点之间的依赖关系或数据流动情况[1]。代码的控制流图反映了代码中语句执行时的跳转流向,而代码的数据依赖图反映了代码中数据的流向。通过将程序表示为图的形式使得模型能够更好地理解代码中不同部分之间的依赖关系。

随着深度学习的发展,基于深度神经网络的图模型被用于代码理解相关任务中,包括代码风格修

正、代码修复、代码注释生成等。Allamanis等人[50]考虑到代码中的长依赖问题，如在代码中变量的定义位置与使用位置之间的距离问题，提出了基于图的代码表征方法，力图学习代码中的语法以及语义结构。该工作首先将代码转换为对应的抽象语法树，之后通过不同的连接规则连接抽象语法树各个节点，获得了不同节点之间的关联关系，这其中就包含了变量之间的依赖关系。最后将构建好的代码图数据作为输入，输入到图神经网络中进行表征学习。该方法在程序变量命名以及变量名误用检测任务中取得了不错的效果。

Lu等人[51]从代码中提取数据流与函数调用信息，将其融合到抽象语法树中，从而将代码构建为一个包含丰富信息的图结构表示。在传统的GGNN（gated graph sequence neural networks）模型上引入了注意力机制，用于获得图中每个节点的重要程度，进而获得更具有区分度的代码表征向量。所生成的代码表征向量用于代码功能分类任务中，实验结果表明，该方法能够有效地应用于代码分类任务中。

Brockschmidt等人[52]同样在代码的抽象语法树上增加相应的边以构建代码图，代码图的构建方法与文献[50]类似。之后采用图神经网络对程序的结构和数据流进行建模完成程序自动生成任务。Ben-Nun等人[53]提出了一种与语言以及平台无关的代码表征方法inst2vec。该方法首先使用编译器对代码进行编译，得到代码的中间表示。但由于该中间表示并没有包含代码之中的数据流信息以及控制流信息，因此该方法将数据流和控制流也融合到该中间表示中，进而构建了代码上下文流图。最后在所构建的图上使用循环神经网络进行建模，获得代码的表征向量。该向量在程序分类实验中准确率取得了当时最好的效果。这些基于图的方法本质上是通过对代码的抽象语法树的改造，显式地将不同节点之间的联系以图的形式表现出来，这一过程能够增加一些人工归纳的规则，因此对一些代码分析任务能够取得较好的效果。

3 代码表征的相关应用

对代码表征的介绍离不开具体的应用场景，对代码表征的质量进行评价通常也是基于具体的代码分析任务。因此，本章将对主要的几个代码分析任务进行简要介绍，并详细介绍代码表征在这些任务中的具体工作。

3.1 代码克隆检测

在软件开发过程中，对代码的修改、复用、变换以及重构会使得软件系统中出现大量的相似甚至重复的代码片段，这个代码复用的过程就被称为代码克隆。已有研究表明，代码克隆这一现象在软件系统中是广泛存在的。如在FreeBSD 5.2.1和Linux 2.6.6系统中，克隆的代码片段分别占比20.4%和22.3%[54-57]。在软件开发过程中，通过复制和粘贴现有的代码虽然可以在一定程度上提高软件开发的速度，但这些克隆的代码同样也对软件系统的维护带来了巨大的挑战[58-64]。当被克隆的代码片段存在缺陷时，该缺陷将会随着其克隆活动的过程传播到软件系统中，将缺陷引入了软件系统中，会对软件系统的稳定性造成破坏，进而提高软件系统的维护成本。而代码克隆检测技术能够帮助软件开发者发现软件系统中相似的代码片段，提升软件系统的稳定性，降低软件系统维护人员的工作量，避免引入与克隆代码相关的缺陷[65]。

目前已有的代码克隆检测方法大多遵循以下思路：(1)首先对代码片段进行预处理；(2)对处理好的代码片段进行信息抽取，将其转换为中间表征；(3)根据表征的方式不同计算不同代码片段之间的相似度，完成克隆检测任务。在这些步骤中，最重要的问题是如何将代码转换为合适的表征，使该表征尽可能多地保留原始代码中所包含的信息，以便进行代码相似度计算。

本节根据上文对代码表征方法的分类介绍，从代码表征的角度，将代码克隆检测方法分为两类：(1)基于代码序列的克隆检测；(2)基于代码结构的克隆检测。根据代码克隆相似程度的不同，Bellon等人将代码克隆分为四种类型[66]，即完全相同的代码（Type-1）、重命名的代码（Type-2）、几乎相同的代码（Type-3）和语义相似的代码（Type-4），从Type-1到Type-4，代码克隆的相似程度逐渐降低，检测的难度也逐渐增加。在普通情况下，仅对代码的文本序列信息进行检测，就能够有效地检测出大部分Type-1、Type-2的代码克隆以及少部分Type-3代码克隆，而对于Type-4类型的代码克隆，则需要基于代码结构的克隆检测方法才能够检测到。以下将对不同的克隆检测方法进行介绍。

基于代码序列的代码克隆检测，将代码视为自然语言文本进行处理。通常使用如后缀树匹配、最长相同序列匹配等匹配算法进行相似度计算。

CCFinder[67]、CP-Miner[2]将原始的代码通过移除空格、注释,进行参数转换等预处理步骤之后,直接在其上进行相似度计算。Duploc[68]将代码视为基本的字符串,通过检测两份代码最长的相同子序列进行相似度比较,从而判断代码的克隆程度。SDD[69]是eclipse开发工具中的一个插件,该方法将代码视为纯文本,通过使用倒排索引和N近邻算法减少克隆检测时间的开销。这种检测方法的优点在于其检测速度快,计算开销小,不依赖于特定的编程语言,甚至对于无法运行的不完整代码也能够进行克隆检测。但其缺点也很明显,由于这类方法仅仅从文本序列的角度出发,没有考虑代码本身所包含的丰富的语法、语义信息,因此对于Type-3、Type-4的代码克隆检测能力不足。还有一些基于代码文本序列的克隆检测方法,如CDSW[70]、XIAO[71]、CCAlingner[15]、srcClone[72]将源代码转换为其对应的哈希值,之后直接使用哈希比较算法计算代码之间的相似程度。与单纯地将代码视为文本的克隆检测方法相比,由于这类方法使用了更符合编译原理的符号序列进行相似度计算,这些方法的克隆检测效果有一定程度的提升。但是,这种检测方法没有考虑代码中所包含的结构信息,因此检测不到代码的结构信息层面上的变化。

基于代码结构的克隆检测方法通常是根据语法规则,将代码转换为对应的抽象语法树,进而进行检测分析。抽象语法树是源代码编译过程中产生的一个中间表示,将源代码中所包含的语法信息以树的形式表现出来,之后采用基于树的相似度匹配算法计算代码之间的相似度。目前,许多工作都尝试从抽象语法树中提取更加丰富有效的代码特征,将抽象语法树转换为不同的表征向量,在这些代码表征向量上进行代码克隆检测。由代码的抽象语法树所生成的代码表征向量,不但包含了代码中的token信息,同时也包含了代码的结构信息、语法规则,因而这类方法具有对Type-3、Type-4克隆类型的检测能力。Zhang等人[10]将代码转换为其对应的抽象语法树,并对抽象语法树进一步划分,得到代码的语句树。通过在语句树级别上的建模,获得了代码表征向量。该表征向量被用于代码克隆检测,检测结果远超于仅使用文本序列的检测方法。但是基于树的方法通常需要遍历树,其计算开销会大于基于代码序列的克隆检测方法。

表2展示的是近五年来基于深度学习的一些代码克隆检测工作。从中可以看出,使用深度学习技术能够有效地进行Type-3、Type-4类型的代码克隆检测。这些方法使用不同的神经网络结构进行代码的特征提取。在此表中所使用的深度学习技术中序列神经网络占比较大,这说明主流的研究方向仍然是将代码作为序列进行处理。但是从此表中也可以看出,目前代码克隆检测模型也仅仅是关注于少部分语言,如Java和C。造成这一现象的主要原因是当前代码克隆检测领域仅有少量的标准数据集。

3.2 代码搜索

随着近几年开源软件和开源社区的发展,越来越多企业和个人在开源社区贡献出软件代码。Singer等人[83]的一项研究发现,软件开发人员在软件开发过程中最频繁的活动之一就是进行代码搜索。当软件开发者在实现某个特定功能的代码需要调用已有的应用程序接口(API)时,如果开发者对其的使

表2 基于深度学习的代码克隆检测方法

Table 2 Deep learning-based code clone detection methods

模型简称	检测类型	神经网络模型	检测语言	时间
CCLearner[73]	Type-1,2,3(ST)	深度神经网络	Java	2017
CDLH[44]	Type-1,2,3,4	长短期记忆网络	Java、C	2017
DeepSim[74]	Type-1,2,3,4	前馈神经网络	Java	2018
ASTNN[10]	Type-1,2,3,4	门控循环单元	Java、C	2019
TECCD[75]	Type-1,2,3	图神经网络	Java	2019
FCCA[76]	Type-1,2,3,4	长短期记忆网络、图神经网络	Java	2020
FCDetector[77]	Type-4	深度神经网络	C	2020
At-biLSTM[78]	Type-1,2,3,4	双向长短期记忆网络	Java、C	2020
Rsharer+[79]	Type-1,2,3,4	卷积神经网络	Java	2020
MISIM[80]	Type-1,2,3,4	图神经网络	C、C++	2020
CodeAli[81]	Type-1,2,3,4	卷积神经网络	Java、C	2021
CACCD[82]	Type-1,2,3,4	双向长短期记忆网络	Java	2021

用方式不太了解，就需要进行代码搜索[11,83-87]。通过在搜索引擎中键入API的名称等信息，可以搜索到该API的具体使用方式。又或者，软件开发者甚至都不清楚哪些API能够实现他所需要完成的功能，也可以通过在搜索引擎中键入功能描述来搜索到可以满足开发需求的API，甚至完整的代码。通常情况下，软件开发者会使用搜索引擎（如百度、谷歌）进行代码的搜索。但这些通用的搜索引擎不是专门针对编程任务所开发的搜索引擎，在搜索过程中不会考虑代码的语义特征，当所需要搜索的代码功能比较复杂时，搜索效果较差。因此，当通用的搜索引擎不能满足开发者的需求时，开发者会在一些专业的代码问答网站（https://sourceforge.net/、https://stackoverflow.com/）或者开源社区（https://www.github.com/）上进行搜索。然而这些网站所返回的搜索结果通常与所想要解决的问题相关度较低，尽管代码片段中包含了查询语句中相关的词汇，但是所返回的结果仍然无法完成开发者的开发需求。

现有的代码搜索技术，通常是将源代码视为纯文本，直接利用信息检索领域中的检索模型去查找并匹配相关的代码片段。这些方法主要是通过比较自然语言查询语句和源代码之间的文本相似性进行搜索。然而因为其仅仅考虑文本相似性，而没有考虑语义或者功能相似性，通常搜索返回的结果不尽人意[86]。随着计算机技术的快速发展，基于数据驱动的机器学习方法得到了广泛的关注。有研究对源代码进行统计分析，发现源代码与自然语言有着相似的统计学规律。因此，对源代码的分析开始借鉴自然语言处理领域的相关方法，如通过计算代码之间的文本表征向量来计算代码之间的相似度，对代码进行摘要提取、代码的跨语言翻译等[88-96]。这种研究方式极大地推动了代码数据分析和挖掘的研究进程。

代码搜索和自然语言处理领域相结合，通常的做法是将自然语言查询语句和源代码转换为同一个向量空间的连续向量，并计算两者的相似程度。这类方式依赖于连续向量所能蕴含的上下文隐含语义信息量的多少。Gu等人[16]提出了CODEnn模型，该模型没有使用传统的文本相似度匹配算法。它将自然语言描述语句和源代码片段共同映射到同一个高维度的向量空间中，使得描述语句和代码具有相似的向量表示。之后，通过计算自然语言查询语句向量和代码向量之间的余弦相似度来返回合适的代码片段。Sachdev等人[97]提出了NCS（neural code search），该方法的主要思想是使用连续向量来获得代码的语义信息。在方法级别上将代码划分为不同的代码片段，并对每个代码片段生成连续的向量嵌入。同时，将自然语言查询也映射到相同的向量空间中，通过计算向量之间的距离来衡量代码之间的相关程度，以供查询。Lv等人[98]提出了CodeHow，该模型是一个包含扩展布尔模型和API匹配的代码搜索工具。首先从代码的在线文档中收集每个API的描述，获得API的描述之后，对其进行编码并计算文本描述与查询之间的相似性，以及API名称与查询之间的相似性。最后组合每个API对之间的相似性，返回与查询匹配的可能相关API。Fang等人[99]与文献[16]采取了相似的做法，将代码拆分成方法名、API、词汇单元三部分，将这三部分转换为对应的向量，并在其上使用自注意力机制，获得各个部分内部元素的注意力分数附加在表征向量中，同时将代码的描述文本也用同样的方式转换为向量，最后使用余弦相似度来衡量代码与代码描述语言之间的相似度，返回代码搜索结果。

有许多研究也关注于代码本身所具有的特殊性，通过将代码转换为对应的抽象语法树，或者程序依赖图，使用深度学习技术获得这些特殊结构所包含的代码信息。与基于文本的嵌入技术类似，这类方法通常将项目中的代码转换为代码图，之后使用图嵌入技术用于表征代码图中节点信息。之后根据代码图解析查询语句，将查询语句的关键词与代码图中的候选节点进行匹配，生成子图并推荐搜索结果。Gu等人[100]首先对代码对应的抽象语法树进行简化，之后在其上引入了树型序列化方法，将抽象语法树转换为词汇单元序列，进行多模态的代码搜索。Meng[101]设计了一种抽象语法树解析算法，能够在将代码转换为向量的过程中保留代码的词法特征和结构特征。Xu等人[102]提出了TabCS模型，在代码的文本特征、代码对应的抽象语法树结构特征以及查询语句上同时使用注意力机制，捕获它们之间的语义相关性。Zou等人[103]通过使用图嵌入技术，将软件项目源代码转换为代码图。这类方法可以有效地表示软件代码图的深层结构信息。

3.3 代码补全

代码补全是自动化软件开发的重要功能之一，能够有效地提高软件开发者的工作效率。代码补全技术基于开发人员的输入，实时预测待补全代码中的类名、方法名、变量名等[104]，通过这种方式能够有

效减轻开发者的键入负担,减少拼写错误,进而提高开发效率。但是,早期的代码补全工具不能很好地满足开发人员的需求,这些工具常常忽略了编程语言的语法规则,所提供的代码补全列表仍然需要进行人工修正。并且,这些代码补全工具通常只利用已有的代码和语法规则,很少考虑待补全代码与上文之间的语义关联。Bruch等人[105]就提出了智能代码补全技术,从已有的代码库中挖掘更多的有效信息用于代码补全[106-120]。

代码补全技术将待补全位置的代码的上下文信息和代码库中学习到的代码上下文信息进行关联,通过计算两者之间的相似度,推荐相似度高的代码片段作为代码补全推荐结果。根据代码补全的对象,可以将代码补全技术分为:(1)标识符补全;(2)代码片段补全;(3)关键词、缩略词补全。标识符补全指的是根据已有的代码,对不完整的标识符进行补全。一般补全的对象包括方法名、变量名、参数名等。代码片段补全指的是对于语句空缺的代码片段,使用代码补全工具自动地生成符合编程语言规范的语句用于补全。关键词、缩略词补全指的是对于输入的简短的词汇,将其补全为完整的函数或参数[121]。但是由于所输入的词汇所能包含的信息量较少,通常情况下这种补全方法的补全结果较差。

虽然对于代码补全技术的研究在近十几年中取得了一定的成果,但是在实际的开发应用中仍然面临以下几个问题需要解决:(1)缺少统一的模型评估指标。目前已有的与代码补全有关的评估指标大多都是机器学习领域的通用指标,如准确率、召回率、平均倒数排名等。目前大部分文献仅采用其中一种指标用于模型的评估,因此对于不同的代码补全方法的性能难以进行比较。(2)缺少真实的基准数据集[122-123]。在机器学习领域,对模型性能的评估不但与评价标准有关,还与所使用的基准数据集有关。目前代码补全的文献中所使用的数据集大多是研究者自行爬取的数据集,缺少一个公认的、合理的、真实的数据集用于评估代码补全模型的效果。

4 分析与讨论

随着近几年嵌入技术在自然语言处理领域中获得的成功,越来越多的代码分析工作也开始采用类似的方法对代码分析领域中尚未解决的问题进行研究。如本文中描述的代码表征研究方向,在目前的代码分析工作中有着重要的地位。其中的词嵌入技术和分布式假说赋予了深度学习工具强大的表征能力。使用神经网络技术,能够将代码所特有的语义信息和结构信息特征转换为计算机擅长处理的实数向量形式。在此过程中无需进行复杂的特征工程分析,减少人为操作所带来的信息损失。但是在现有的代码表征工作中,仍然存在以下几点困难尚未解决,包括代码库中存在的超出词表问题、表征模型的构建方式以及模型所生成的代码表征的质量评价标准等。接下来本章将对这些问题进行详细分析。

4.1 代码库中的超出词表问题

当前代码分析文献大都根据代码文本中所存在的自然性这一特征进行具体的代码分析任务,如代码补全、代码克隆检测这类任务都是将代码视为自然语言文本进行处理。如本文背景知识部分所述,当使用自然语言技术对代码文本进行处理时,由于软件开发人员在编写代码时可以自由地创建标识符,这一过程将会产生一个规模巨大且稀疏的代码词表。代码词表的规模会直接影响代码分析任务的效率。在神经网络模型训练的过程中,通常的做法是对这个大词表中的词汇单元个数进行数量的限制,因此当某个词汇在词表中没有出现过,那么神经网络模型将无法对其进行处理。这就是代码分析中的超出词表问题(OoV)。

在自然语言处理领域,对于超出词表问题已有很多相关文献[124-125]。这些方法将完整的词汇单元拆分为能够组合成完整词汇的子词汇单元,在自然语言分析领域中取得了一定的成果。在代码分析领域中,同样也有对超出词表问题进行研究的一些工作。文献[126],该工作本质上是使用当前自然语言处理领域中字节对编码(byte pair encoder, BPE)方法对大型代码库进行分析,并未考虑代码本身所具有的特性,如代码的局部重复性、标识符的整体性。文献[127]研究了代码中的超出词表问题对之后的机器学习模型性能的影响,发现词表的大小、输入数据中超出词表词汇的占比对机器学习模型的影响是很大的。文献[118]对代码词汇单元的拆分方式如图4所示,若是将setter变量名拆分为set、ter两部分,将直接丢失其作为一个完整变量的信息。

因此,如何处理代码的超出词表问题,减轻其对后续代码分析任务的影响亦是一个重点的研究方向。本文认为可以从以下几点进行研究:(1)衡量词库中词汇单元的信息量,剔除信息量小的词汇单元;(2)构建特征提取能力强的神经网络模型,从字符层

Java code:

```
public AttributeContext(Method setter, Object value) {
    this.value = value;
    this.setter = setter;
}
```

Subword units:

```
public</t> Attribute Con text</t> (</t> Method</t> set ter</t> ,</t> Object</t>
value</t> )</t> {</t> this</t> .</t> value</t> =</t> value</t> ;</t> this</t>
.</t> set ter</t> =</t> set ter</t> ;</t> }</t>
```

图4 Java代码的词汇单元与子词汇单元

Fig.4 Word and subword units of Java code

面对代码信息进行提取。

4.2 表征模型的构建

本文之前所列举的代码表征工作中，均是以结合具体的代码分析任务进行介绍。这是因为这些工作均是以具体的代码分析任务进行表征模型的设计。如代码克隆检测、代码分类这种依赖于代码模式特征的任务。对代码补全任务，应尽可能多地提取代码的结构信息。对于代码迁移任务，则通常需要采用序列神经网络对代码进行建模。在自然语言分析领域中，存在诸如BERT（bidirectional encoder representation from transformers）[128]、GPT（generative pre-training）[129]、ELMo（embeddings from language models）[130]这类通用的预训练模型，能够直接应用于大多数的自然语言分析任务。而在代码分析领域中，尚缺少对于预训练模型应用于代码分析任务的工作。现有的生成代码表征的模型通常都是有监督学习模型，即所使用代码表征模型都是基于具体的代码分析任务构建的。这从根本上使得所学习得到的代码表征向量不能适用于多种任务。如目前在代码表征领域影响较高的Code2vec工作，其生成的代码表征向量也被证明不能直接简单地应用于代码分析任务中[131]。对于如何构造一个具有更强泛化能力的代码表征模型本文有以下几个观点：（1）将传统的代码分析工作中代码特征提取部分融合进代码表征模型的构建过程。（2）增加模型训练的数据量，提高代码表征模型的特征提取能力。

4.3 代码表征质量的评价标准

代码实际上具有高度的抽象性和复杂性，代码的大部分性质难以用简单明了的数学语言来定义。因此，对于代码分析模型所生成的代码表征的评价就显得尤为重要。但是目前对代码表征质量的评估并没有一个统一的标准。现有的工作都是基于具体的分析任务对表征向量进行评价。甚至在同一种类型的分析任务中，都存在着多种不同的指标。如在代码克隆检测的相关工作中，所使用的评价指标包括精确率（precision）、召回率（recall）、F1值、准确率（accuracy）等评价指标，同时代码检测任务中所使用的数据集包括BigCloneBench[43]、GoogleCodeJam[74]、OnlineJudge[19]等不同形式的数据集。在代码搜索任务中，也是通过计算最后推荐结果的归一化折损累计增益（normalized discounted cumulative gain, NDCG）来衡量代码表征质量的好坏。这种基于具体任务的结果来判断代码表征质量优劣的方式，并不是直接对表征向量的质量进行衡量。同时在代码分析任务中，不同任务所使用的数据集是不同的，也存在着许多工作是基于研究者自己所标注的数据，这使得应用于不同代码分析任务中的表征模型不能在同一标准上进行比较。对于此问题，本文建议以众包的形式构建代码分析领域通用高质量数据集，并在统一的评价标准上进行方法比较。

5 结束语

代码表征在软件分析任务中占据了重要的地位。通过使用深度学习模型生成代码表征向量，自动地提取代码中所包含的隐含特征，降低对人工制定特征的依赖，进而提升代码分析任务的效率。本文介绍了代码表征的基本概念，并对近几年来不同的代码表征及其应用的工作进行了综述，对这些工作的原理与技术进行了分类与总结，最后分析并讨论了现有的代码表征工作中仍然存在的问题以及对应的解决方法。

参考文献：

[1] 刘芳, 李戈, 胡星. 基于深度学习的程序理解研究进展[J]. 计算机研究与发展, 2019, 56(8): 1605-1620.
LIU F, LI G, HU X. Program comprehension based on deep learning[J]. Journal of Computer Research and Development, 2019, 56(8): 1605-1620.

[2] HINDLE A, BARR E T, SU Z, et al. On the naturalness of software[C]//Proceedings of the 2012 34th International Conference on Software Engineering, Zurich, Jun 2-9, 2012. Washington: IEEE Computer Society, 2012: 837-847.

[3] ROBBES R, LANZA M. How program history can improve code completion[C]//Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, L'Aquila, Sep 15-19, 2008. Washington: IEEE Computer Society, 2008: 317-326.

[4] PROKSCH S, LERCH J, MEZINI M. Intelligent code completion with Bayesian networks[J]. ACM Transactions on

Software Engineering and Methodology, 2015, 25(1): 1-31.

[5] BIELIK P, RAYCHEV V, VECHEV M T. PHOG: probabilistic model for code[C]//Proceedings of the 33rd International Conference on Machine Learning, New York, Jun 19-24, 2016: 2933-2942.

[6] OMORI T, KUWABARA H, MARUYAMA K. A study on repetitiveness of code completion operations[C]//Proceedings of the 2012 28th IEEE International Conference on Software Maintenance, Trento, Sep 23-28, 2012. Washington: IEEE Computer Society, 2012: 584-587.

[7] TU Z P, SU Z D, DEVANBU P T. On the localness of software[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, China, Nov 16-22, 2014. New York: ACM, 2014: 269-280.

[8] OSCAR K. TF-IDF inspired detection for cross-language source code plagiarism and collusion[J]. Computer Science, 2020, 21: 113-134.

[9] LE T H, CHEN H, BABAR M A. Deep learning for source code modeling and generation: models, applications, and challenges[J]. ACM Computing Surveys, 2020, 53(3): 1-38.

[10] ZHANG J, WANG X, ZHANG H Y. A novel neural source code representation based on abstract syntax tree[C]//Proceedings of the 41st International Conference on Software Engineering, Montreal, May 25-31, 2019. Piscataway: IEEE, 2019: 783-794.

[11] 刘斌斌, 董威, 王戟. 智能化的程序搜索与构造方法综述[J]. 软件学报, 2018, 29(8): 2180-2197.
LIU B B, DONG W, WANG J. Survey on intelligent search and construction methods of program[J]. Journal of Software, 2018, 29(8): 2180-2197.

[12] WHITE M, TUFANO M, VENDOME C. Deep learning fragments for code clone detection[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Singapore, Sep 3-7, 2016. New York: ACM, 2016: 87-98.

[13] WHITE M, VENDOME C. Toward deep learning software repositories[C]//Proceedings of the 12th IEEE/ACM Working Conference on Mining Software Repositories, Florence, May 16-17, 2015. Washington: IEEE Computer Society, 2015: 334-345.

[14] WHITE M, TUFANO M, MARTINEZ M, et al. Sorting and transforming program repair ingredients via deep learning code similarities[C]//Proceedings of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, Hangzhou, Feb 24-27, 2019. Piscataway: IEEE, 2019: 479-490.

[15] WANG P P, SVAJLENKO J, WU Y Z, et al. CCAligner: a token based large-gap clone detector[C]//Proceedings of the 40th International Conference on Software Engineering, Gothenburg, May 27-Jun 3, 2018. New York: ACM, 2018: 1066-1077.

[16] GU X D, ZHANG H Y, KIM S H. Deep code search[C]//Proceedings of the 40th International Conference on Software Engineering, Gothenburg, May 27-Jun 3, 2018. New York: ACM, 2018: 933-944.

[17] ALON U, ZILBERSTEIN M, LEVY O. A general path-based representation for predicting program properties[C]//Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, Philadelphia, Jun 18-22, 2018. New York: ACM, 2018: 404-419.

[18] ALON U, ZILBERSTEIN M, LEVY O, et al. Code2vec: learning distributed representations of code[C]//Proceedings of the 2019 ACM on Programming Languages, Cascais, Jan 13-19, 2019. New York: ACM, 2019: 1-29.

[19] MOU L L, LI G, ZHANG L. Convolutional neural network over tree structures for programming language processing [C]//Proceedings of the 30th AAAI Conference on Artificial Intelligence, Phoenix, Feb 12-17, 2016. Menlo Park: AAAI, 2016: 1287-1293.

[20] BÜCH L, ANDRZEJAK A. Learning-based recursive aggregation of abstract syntax trees for code clone detection[C]//Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering, Hangzhou, Feb 24-27, 2019. Piscataway: IEEE, 2019: 95-104.

[21] SAHLGREN M. The word-space model: using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces [D]. Stockholm: Institutionen för Lingvistik, 2006.

[22] DUMAIS S T. Latent semantic analysis[J]. Annual Review of Information Science and Technology, 2004, 38(1): 188-230.

[23] BLEI D M, NG A Y, JORDAN M I. Latent Dirichlet allocation[J]. Journal of Machine Learning Research, 2003, 3(1): 993-1022.

[24] ŘEHŮŘEK R, SOJKA P. Software framework for topic modelling with large corpora[C]//Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Malta, May 22, 2010. Valletta: University of Malta, 2004: 45-50.

[25] LE Q V, MIKOLOV T. Distributed representations of sentences and documents[C]//Proceedings of the 31st International Conference on Machine Learning, Beijing, Jun 21-26, 2014: 1188-1196.

[26] 蹇松雷. 基于复杂异构数据的表征学习研究[D]. 长沙: 国防科技大学, 2019.
JIAN S L. Research on the representation learning of com-

plex heterogeneous data[D]. Changsha: National University of Defense Technology, 2019.

[27] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient estimation of word representations in vector space[J]. arXiv: 1301.3781,2013.

[28] KAUR A, NAYYAR R. A comparative study of static code analysis tools for vulnerability detection in C/C ++ and JAVA source code[J]. Procedia Computer Science, 2020, 171: 2023-2029.

[29] HARER J, KIM L, RUSSELL R, et al. Automated software vulnerability detection with machine learning[J]. arXiv:1803. 04497, 2018.

[30] CHEN Z M, MONPERRUS M. The remarkable role of similarity in redundancy-based program repair[J]. arXiv:1811. 05703, 2018.

[31] HENKEL J, LAHIRI S, LIBLIT B, et al. Code vectors: understanding programs through embedded abstracted symbolic traces[C]//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista, Nov 4-9, 2018. New York: ACM, 2018: 163-174.

[32] NGUYEN T D, NGUYEN A T, PHAN H D, et al. Exploring API embedding for API usages and applications[C]// Proceedings of the 39th International Conference on Software Engineering, Buenos Aires, May 20- 28, 2017. Piscataway: IEEE, 2017: 438-449.

[33] PRADEL M, SEN K. DeepBugs: a learning approach to name-based bug detection[J]. Proceedings of the ACM on Programming Languages, 2018, 2: 1-25.

[34] IYER S, KONSTAS I, CHEUNG A. Summarizing source code using a neural attention model[C]//Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, Berlin, Aug 7-12, 2016. Stroudsburg: ACL, 2016: 2073-2083.

[35] ALLAMANIS M, PENG H, SUTTON C. A convolutional attention network for extreme summarization of source code [C]//Proceedings of the 33rd International Conference on Machine Learning, New York, Jun 19-24, 2016: 2091-2100.

[36] LI J, WANG Y, LYU M R, et al. Code completion with neural attention and pointer networks[J]. arXiv:1711.09573, 2017.

[37] BHOOPCHAND A, ROCKSTASCHEL T, BARR E. Learning python code suggestion with a sparse pointer network [J]. arXiv:1611.08307,2016.

[38] SHUAI J, XU L, LIU C, et al. Improving code search with co-attentive representation learning[C]//Proceedings of the 28th International Conference on Program Comprehension, Seoul, Jul 13-15, 2020. New York: ACM, 2020: 196-207.

[39] GU X D, ZHANG H Y, ZHANG D M, et al. Deep API learning[C]//Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Seattle, Nov 13-18, 2016. New York: ACM, 2016: 631-642.

[40] LU X F, JIANG F S, ZHOU X, et al. ASSCA: API sequence and statistics features combined architecture for malware detection[J]. Computer Networks, 2019, 157: 99-111.

[41] SAIFULLAH C M. Learning APIs through mining code snippet examples[D]. Saskatoon: University of Saskatchewan, 2020.

[42] HU X, LI G, XIA X. Summarizing source code with transferred API knowledge[C]//Proceedings of the 27th International Joint Conference on Artificial Intelligence, Stockholm, Jul 13-19, 2018: 2269-2275.

[43] SVAJLENKO J, ISLAM J F, KEIVANLOO I, et al. Towards a big data curated benchmark of inter-project code clones[C]//Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, Sep 29-Oct 3, 2014. Washington: IEEE Computer Society, 2014: 476-480.

[44] WEI H H, LI M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code[C]//Proceedings of the 26th International Joint Conference on Artificial Intelligence, Melbourne, Aug 19-25, 2017: 3034-3040.

[45] CHEN L, YE W, ZHANG S K. Capturing source code semantics via tree-based convolution over API-enhanced AST[C]//Proceedings of the 16th ACM International Conference on Computing Frontiers, Alghero, Apr 30- May 2, 2019. New York: ACM, 2019: 174-182.

[46] WANG W H, LI G, MA B, et al. Detecting code clones with graph neural network and flow-augmented abstract syntax tree[C]//Proceedings of the 27th IEEE International Conference on Software Analysis, Evolution and Reengineering, London, Feb 18-21, 2020. Piscataway: IEEE, 2020: 261-271.

[47] HU X, LI G, XIA X. Deep code comment generation[C]// Proceedings of the 26th Conference on Program Comprehension, Gothenburg, May 27-28, 2018. New York: ACM, 2018: 200-210.

[48] ALON U, BRODY S, LEVY O, et al. Code2seq: generating sequences from structured representations of code[J]. arXiv: 1808.01400,2018.

[49] ALON U, SADAKA R, LEVY O, et al. Structural language models of code[C]//Proceedings of the 2020 International Conference on Machine Learning. New York: ACM, 2020: 245-256.

[50] ALLAMANIS M, BROCKSCHMIDT M, KHADEMI M. Learning to represent programs with graphs[J]. arXiv: 1711.00740, 2017.

[51] LU M M, TAN D W, XIONG N X, et al. Program classification using gated graph attention neural network for online programming service[J]. arXiv:1903.03804, 2019.

[52] BROCKSCHMIDT M, ALLAMANIS M, GAUNT A L. Generative code modeling with graphs[J]. arXiv:1805.08490, 2018.

[53] BEN-NUN T, JAKOBOVITS A S, HOEFLER T. Neural code comprehension: a learnable representation of code semantics[C]//Proceedings of the 32nd International Conference on Neural Information Processing Systems, Montréal, Dec 3-8, 2018: 3589-3601.

[54] LI Z M, LU S, MYAGMAR S, et al. CP-Miner: finding copy-paste and related bugs in large-scale software code[J]. IEEE Transactions on Software Engineering, 2006, 32(3): 176-192.

[55] CHEN W K, LI B G, GUPTA R. Code compaction of matching single-entry multiple-exit regions[C]//LNCS 2694: Proceedings of the 10th International Symposium Static Analysis. Berlin, Heidelberg: Springer, 2003: 401-417.

[56] KIM M, SAZAWAL V, NOTKIN D, et al. An empirical study of code clone genealogies[C]//Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Lisbon, Sep 5-9, 2005. New York: ACM, 2005: 187-196.

[57] PATENAUDE J, MERLO E, DAGENAIS M, et al. Extending software quality assessment techniques to Java systems[C]//Proceedings of the 7th International Workshop on Program Comprehension, Pittsburgh, May 5-7, 1999. Washington: IEEE Computer Society, 1999: 49-56.

[58] SHENEAMER A, KALITA J. A survey of software clone detection techniques[J]. International Journal of Computer Applications, 2016, 137(10): 1-21.

[59] BAKER B. On finding duplication and near-duplication in large software systems[C]//Proceedings of the 2nd Working Conference on Reverse Engineering, Toronto, Jul 14-16, 1995. Piscataway: IEEE, 1995: 86-95.

[60] ROY C K, CORDY J R. NICAD: accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization[C]//Proceedings of the 16th IEEE International Conference on Program Comprehension, Amsterdam, Jun 10-13, 2008. Washington: IEEE Computer Society, 2008: 172-181.

[61] MONDAL M, RAHMAN M S, ROY C K, et al. Is cloned code really stable[J]. Empirical Software Engineering, 2018, 23(2): 693-770.

[62] JÜRGENS E, DEISSENBOECK F, HUMMEL B, et al. Do code clones matter[C]//Proceedings of the 31st International Conference on Software Engineering, Vancouver, May 16-24, 2009. Piscataway: IEEE, 2019: 485-495.

[63] MONDAL M, ROY C, SCHNEIDER K. Dispersion of changes in cloned and non-cloned code[C]//Proceeding of the 6th International Workshop on Software Clones, Zurich, Jun 4, 2012. Washington: IEEE Computer Society, 2012: 29-35.

[64] LOZANO A, WERMELINGER M. Tracking clones' imprint [C]//Proceeding of the 4th ICSE International Workshop on Software Clones, Cape Town. New York: ACM, 2010: 65-72.

[65] 陈秋远, 李善平, 鄢萌, 等. 代码克隆检测研究进展[J]. 软件学报, 2019, 30(4): 962-980.
CHEN Q Y, LI S P, YAN M, et al. Code clone detection: a literature review[J]. Journal of Software, 2019, 30(4): 962-980.

[66] BELLON S, KOSCHKE R, ANTONIOL G, et al. Comparison and evaluation of clone detection tools[J]. IEEE Transactions on Software Engineering, 2007, 33(9): 577-591.

[67] 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.

[68] DUCASSE S, RIEGER M, DEMEYER S. A language independent approach for detecting duplicated code[C]// Proceedings of the 1999 International Conference on Software Maintenance, Oxford, Aug 30-Sep 3, 1999. Washington: IEEE Computer Society, 1999: 109-118.

[69] LEE S, JEONG I. SDD: high performance code clone detection system for large scale source code[C]//Proceedings of the Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, San Diego, Oct 16-20, 2005. New York: ACM, 2005: 140-141.

[70] MURAKAMI H, HOTTA K, HIGO Y, et al. Gapped code clone detection with lightweight source code analysis[C]// Proceedings of the IEEE 21st International Conference on Program Comprehension, San Francisco, May 20-21, 2013. Washington: IEEE Computer Society, 2013: 93-102.

[71] DANG Y N, ZHANG D M, GE S, et al. XIAO: tuning code clones at hands of engineers in practice[C]//Proceedings of the 28th Annual Computer Security Applications, Orlando, Dec 3-7, 2012. New York: ACM, 2012: 369-378.

[72] ALOMARI H, MATTHEW S. Clone detection through srcClone: a program slicing based approach[J]. Journal of Systems and Software, 2022, 184: 111115.

[73] LI L, FENG H, ZHUANG W. CCLearner: a deep learning-

based clone detection approach[C]//Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution, Shanghai, Sep 17-22, 2017. Washington: IEEE Computer Society, 2017: 249-260.

[74] ZHAO G, HUANG J. DeepSim: deep learning code functional similarity[C]//Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista, Nov 4-9, 2018. New York: ACM, 2018: 141-151.

[75] GAO Y, WANG Z, LIU S. TECCD: a tree embedding approach for code clone detection[C]//Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution, Cleveland, Sep 29-Oct 4, 2019. Piscataway: IEEE, 2019: 145-156.

[76] HUA W, SUI Y, WAN Y. FCCA: hybrid code representation for functional clone detection using attention networks[J]. IEEE Transactions on Reliability, 2020, 70(1): 304-318.

[77] FANG C, LIU Z, SHI Y, et al. Functional code clone detection with syntax and semantics fusion learning[C]// Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2020: 516-527.

[78] MENG Y, LIU L. A deep learning approach for a source code detection model using self-attention[J]. Complexity, 2020, 9: 1-15.

[79] GUO C, YANG H, HUANG D. Review sharing via deep semi-supervised code clone detection[J]. IEEE Access, 2020, 8: 24948-24965.

[80] YE F, ZHOU S, VENKAT A. MISIM: an end-to-end neural code similarity system[J]. arXiv:2006.05265,2020.

[81] ZHANG A, LIU K, FANG L, et al. Learn to align: a code alignment network for code clone detection[C]//Proceedings of the 28th Asia-Pacific Software Engineering Conference, Taipei, China, Dec 6-9, 2021. Piscataway: IEEE, 2021: 1-11.

[82] LIANG H, AI L. AST-path based compare-aggregate network for code clone detection[C]//Proceedings of the 2021 International Joint Conference on Neural Networks, Shenzhen, Jul 18-22, 2021. Piscataway: IEEE, 2021: 1-8.

[83] SINGER J, LETHB T C, VINSON N G, et al. An examination of software engineering work practices[C]//Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research, Toronto, Nov 10-13: 21.

[84] ZHONG H, XIE T, ZHANG L, et al. mAPO: mining and recommending API usage patterns[C]//LNCS 5653: Proceedings of the 23rd European Conference on Object-Oriented Programming, Genoa, Jul 6-10, 2009. Berlin, Heidelberg: Springer, 2009: 318-343.

[85] 张峰逸, 彭鑫, 陈驰. 基于深度学习的代码分析研究综述[J]. 计算机应用与软件, 2018, 35(6): 9-17.
ZHANG F Y, PENG X, CHEN C. Research on code analysis based on deep learning[J]. Computer Applications and Software, 2018, 35(6): 9-17.

[86] SUBRAMANIAN S, INOZEMTSEVA L, HOLMES R. Live API documentation[C]//Proceedings of the 36th International Conference on Software Engineering, Hyderabad, May 31-Jun 7, 2014. New York: ACM, 2014: 643-652.

[87] KIM K, KIM D, BISSYANDÉ T F, et al. FaCoY: a code-to-code search engine[C]//Proceedings of the 40th International Conference on Software Engineering, Gothenburg, May 27-Jun 3, 2018. New York: ACM, 2018: 946-957.

[88] DEERWESTER S C, DUMAIS S T, LANDAUER T K, et al. Indexing by latent semantic analysis[J]. Journal of the American Society for Information Science, 1990, 41(6): 391-407.

[89] BENGIO Y, DUCHARME R, VINCENT P, et al. A neural probabilistic language model[J]. Journal of Machine Learning Research, 2003, 3(2): 1137-1155.

[90] EGOZI O, MARKOVITCH S, GABRILOVICH E. Concept-based information retrieval using explicit semantic analysis [J]. ACM Transactions on Information Systems, 2011, 29 (2): 1-34.

[91] SCHUHMACHER M, PONZETTO S P. Knowledge-based graph document modeling[C]//Proceedings of the 7th ACM International Conference on Web Search and Data Mining, New York, Feb 24-28, 2014. New York: ACM, 2014: 543-552.

[92] LIU X T, FANG H. Latent entity space: a novel retrieval approach for entity-bearing queries[J]. Information Retrieval Journal, 2015, 18(6): 473-503.

[93] XIONG C Y, CALLAN J. EsdRank: connecting query and documents through external semi-structured data[C]//Proceedings of the 24th ACM International Conference on Information and Knowledge Management, Melbourne, Oct 19-23, 2015. New York: ACM, 2015: 951-960.

[94] RAVIV H, KURLAND O, CARMEL D. Document retrieval using entity-based language models[C]//Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, Pisa, Jul 17-21, 2016. New York: ACM, 2016: 65-74.

[95] NI Y, XU Q K, CAO F, et al. Semantic documents relatedness using concept graph representation[C]//Proceedings of the 9th ACM International Conference on Web Search and Data Mining, San Francisco, Feb 22-25, 2016. New York: ACM, 2016: 635-644.

[96] GABRILOVICH E, MARKOVITCH S. Computing semantic relatedness using Wikipedia-based explicit semantic analysis

[C]//Proceedings of the 2007 International Joint Conference on Artificial Intelligence, Hyderabad, Jan 6-12, 2007. San Mateo: Morgan Kaufmann, 2007: 1606-1611.

[97] SACHDEV S, LI H Y, LUAN S F, 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, Philadelphia, Jun 18-22, 2018. New York: ACM, 2018: 31-41.

[98] LV F, ZHANG H Y, LOU J G, et al. CodeHow: effective code search based on API understanding and extended Boolean model[C]//Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, Lincoln, Nov 9-13, 2015. Washington: IEEE Computer Society, 2015: 260-270.

[99] FANG S, TAN Y, ZHANG T, et al. Self-attention networks for code search[J]. Information and Software Technology, 2021, 134: 106542-106553.

[100] GU J, CHEN Z, MONPERRUS M. Multimodal representation for neural code search[C]//Proceedings of the 2021 International Conference on Software Maintenance and Evolution, Luxembourg, Sep 27-Oct 1, 2021. Piscataway: IEEE, 2021: 483-494.

[101] MENG Y. An intelligent code search approach using hybrid encoders[J]. Wireless Communications and Mobile Computing, 2021: 9990988.

[102] XU L, YANG H, LIU C, et al. Two-stage attention-based model for code search with textual and structural features [C]//Proceedings of the 28th IEEE International Conference on Software Analysis, Evolution and Reengineering, Honolulu, Mar 9-12, 2021. Piscataway: IEEE, 2021: 342-353.

[103] ZOU Y Z, LING C Y, LIN Z Q, et al. Graph embedding based code search in software project[C]//Proceedings of the 10th Asia-Pacific Symposium on Internetware, Beijing, Sep 16, 2018. New York: ACM, 2018: 1-10.

[104] GORIN R E. SPELL: a spelling checking and correction program[J]. Online Documentation for the DEC-10 Computer, 1971: 147-160.

[105] BRUCH M, MONPERRUS M, MEZINI M. Learning from examples to improve code completion systems[C]// Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Amsterdam, Aug 24-28, 2009. New York: ACM, 2009: 213-222.

[106] HOU D Q, PLETCHER D M. An evaluation of the strategies of sorting, filtering, and grouping API methods for code completion[C]//Proceedings of the IEEE 27th International Conference on Software Maintenance, Williamsburg, Sep 25-30, 2011. Washington: IEEE Computer Society, 2011: 233-242.

[107] LEE Y Y, HARWELL S, Khurshid S, et al. Temporal code completion and navigation[C]//Proceedings of the 35th International Conference on Software Engineering, San Francisco, May 18-26, 2013. Washington: IEEE Computer Society, 2013: 1181-1184.

[108] NGUYEN A T, NGUYEN H A, NGUYEN T T, et al. GraPacc: a graph-based pattern-oriented, context-sensitive code completion tool[C]//Proceedings of the 34th International Conference on Software Engineering, Zurich, Jun 2-9, 2012. Washington: IEEE Computer Society, 2012: 1407-1410.

[109] JIN X H, SERVANT F. The hidden cost of code completion: understanding the impact of the recommendation-list length on its efficiency[C]//Proceedings of the 15th International Conference on Mining Software Repositories, Gothenburg, May 28-29, 2018. New York: ACM, 2018: 70-73.

[110] ZHONG H, WANG X Y. Boosting complete-code tool for partial program[C]//Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, Urbana, Oct 30-Nov 3, 2017. Washington: IEEE Computer Society, 2017: 671-681.

[111] 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, Saint Petersbury, Aug 18-26, 2013. New York: ACM, 2013: 532-542.

[112] DE SOUZA AMORIM L E, ERDWEG S, WACHSMUTH G, et al. Principled syntactic code completion using placeholders[C]//Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, Amsterdam, Oct 31-Nov 1, 2016. New York: ACM, 2016: 163-175.

[113] HOU D Q, PLETCHER D M. Towards a better code completion system by API grouping, filtering, and popularity-based ranking[C]//Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, Cape Town, May 4, 2010. New York: ACM, 2010: 26-30.

[114] JACOBELLIS J, MENG N, KIM M. Cookbook: in Situ code completion using edit recipes learned from examples [C]//Companion Proceedings of the 36th International Conference on Software Engineering, Hyderabad, May 31-Jun 7, 2014. New York: ACM, 2014: 584-587.

[115] NGUYEN T T, PHAM H V, VU P M, et al. Recommending API usages for mobile Apps with hidden Markov model

[C]//Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, Lincoln, Nov 9-13, 2015. Washington: IEEE Computer Society, 2015: 795-800.

[116] GVERO T, KUNCAK V, KURAJ I, et al. Complete completion using types and weights[J]. ACM SIGPLAN Notices, 2013, 48(6): 27-38.

[117] FERNANDES P, ALLAMANIS M, BROCKSCHMIDT M. Structured neural summarization[J]. arXiv:1811.01824,2018.

[118] KARAMPATSIS R, BABII H, ROBBES R, et al. Big code !=big vocabulary: open-vocabulary models for source code[C]//Proceedings of the 42nd International Conference on Software Engineering, Seoul, Jun 27-Jul 19, 2020. New York: ACM, 2020: 1073-1085.

[119] 杨博, 张能, 李善平, 等. 智能代码补全研究综述[J]. 软件学报, 2020, 31(5): 1435-1453.
YANG B, ZHANG N, LI S P, et al. Survey of intelligent code completion[J]. Journal of Software, 2020, 31(5): 1435-1453.

[120] HAN S, WALLACE D R, MILLER R C. Code completion of multiple keywords from abbreviated input[J]. Automated Software Engineering, 2011, 18(3/4): 363-398.

[121] HAN S, WALLACE D R, MILLER R C. Code completion from abbreviated input[C]//Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering, Auckland, Nov 16-20, 2009. Washington: IEEE Computer Society, 2009: 332-343.

[122] RAYCHEV V, BIELIK P, VECHEV M. Probabilistic model for code with decision trees[J]. ACM SIGPLAN Notices, 2016, 51(10): 731-747.

[123] HELLENDOORN V J, DEVANBU P. Are deep neural networks the best choice for modeling source code?[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Sep 4-8, 2017. New York: ACM, 2017: 763-773.

[124] BAZZI I. Modelling out-of-vocabulary words for robust speech recognition[D]. Massachusetts Institute of Technology, 2002.

[125] LUONG M T, SOCHER R, MANNING C D. Better word representations with recursive neural networks for morphology[C]//Proceedings of the 17th Conference on Computational Natural Language Learning, Sofia, Aug 8-9, 2013. Stroudsburg: ACL, 2013: 104-113.

[126] HARRIS Z. Distributional structure[J]. Word, 1981, 10(2/3): 146-162.

[127] BABII H, JANES A, ROBBES R. Modeling vocabulary for big code machine learning[J]. arXiv:1904.01873,2019.

[128] DEVLIN J, CHANG M W, LEE K. BERT: PRE-training of deep bidirectional transformers for language understanding[J]. arXiv:1810.04805, 2018.

[129] RADFORD A, NARASIMHAN K, SALIMANS T. Improving language understanding by generative pre-training [EB/OL]. [2021-07-06]. https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

[130] PETERS M, NEUMANN M, IYYER M. Deep contextualized word representations[J]. arXiv:1802.05365,2018.

[131] KANG H J, BISSYANDÉ T F, LO D. Assessing the generalizability of code2vec token embeddings[C]//Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, San Diego, Nov 11-15, 2019. Piscataway: IEEE, 2019: 1-12.

张祥平(1993—),男,福建三明人,博士研究生,主要研究方向为代码表征、代码克隆检测。
ZHANG Xiangping, born in 1993, Ph.D. candidate. His research interests include code representation and code clone detection.

刘建勋(1970—),男,湖南衡阳人,博士,教授,主要研究方向为服务计算、云计算。
LIU Jianxun, born in 1970, Ph.D., professor. His research interests include service computing and cloud computing.