文章编号：2095-2163(2021)07-0216-06 中图分类号：TP311 文献标志码：A

基于程序切片和 BiGRU 的代码搜索

杨 腾，赵逢禹，刘 亚
（上海理工大学 光电信息与计算机工程学院，上海 200093）

摘 要：代码复用可以有效缩短软件开发的时间，而代码搜索是代码复用的主要途径。提出了一种基于程序切片和 BiGRU 的代码搜索方法，该方法通过构建源代码的程序依赖图，以程序依赖图中出度最大的节点作为兴趣点构建前向切片。将程序切片与源代码的其他相关特征一起构成代码特征。把代码特征和代码的功能描述通过嵌入模块输入到 BiGRU 网络中，结合注意力机制训练 BiGRU 模型。用户输入功能查询语句，模型返回向量值最接近的代码。为了验证该模型的可行性和有效性，从开源代码库下载了 Java 项目，构建了数据集并进行实验。实验结果表明，提出的基于程序切片和 BiGRU 的方法在代码搜索的准确率和相关性排名等方面都有所改进。

关键词：代码搜索；程序切片；BiGRU；注意力机制

Code search based on program slicing and BiGRU

YANG Teng, ZHAO Fengyu, LIU Ya
(School of Optical-Electrical & Computer Engineering, University of Shanghai for Science & Technology, Shanghai 200093, China)

【Abstract】Code reuse can effectively shorten the time of software development, and code search is the main way of code reuse. This paper proposes a code search method based on program slicing and BiGRU. The method builds a program dependency graph of source code, and takes the node with the highest degree in the program dependency graph as the point of interest to construct program slices. The features of source code are formed by combining program slices with other relevant method features. Code features and code function descriptions are input into the BiGRU network through an embedding module, and BiGRU network model is generated by combining attention mechanism. Users enter the function query statement, and the model returns the code with the closest vector value. In order to verify the feasibility and effectiveness of the model, several Java projects are downloaded from the open source library to build the data set and conduct experiments. Experimental results show that the proposed method based on program slicing and BiGRU has some improvements in code search accuracy and relevance ranking.

【Key words】code search; program slice; BiGRU; attention

0 引 言

重用现有的代码片段可以有效缩短软件开发的时间，开发人员可以通过代码搜索重用已有的方法，提高开发效率。研究表明，代码搜索是软件开发人员使用最频繁的活动之一[1]。近年来，随着开源理念的加强，开源项目的质量和数量在不断地增加，开源代码库为人们提供了丰富的资源，但如何快速准确地搜索到需要的代码成为了一个新的问题。

早期的代码搜索方法通常采用全文检索技术，通过全文检索获得与查询关键词匹配的代码段。基于信息检索的方法通常将源代码视为纯文本，并利用信息检索模型来检索与给定查询关键字匹配的相关代码段。Sindhgatta[2] 提出了基于 Lucene 的 JSearch，该工具通过结构化查询语言搜索 Java 文件。Hill 等人[3] 提出了基于词组的搜索技术，该技术基于关键字搜索，并且还引入了基于短语概念的评分机制，通过查询单词的位置、语义角色、头部距离和使用信息对程序的相关性进行评分。Bajracharya 等人[4] 提出了 Sourcerer，Sourcerer 以查询语句为输入，通过分析代码中 API 调用情况，搜索相似的代码实体，根据这些实体的名称进行扩展，在此基础上使用 TF-IDF 技术搜索流行的类。Raghothaman 等人[5] 提出了 SWIM，与之前的信息检索技术不同，该方法使用词袋模型，根据查询找到相应的 API，再把 API 组合成代码段返回给用户。

基金项目：“十三五”密码发展基金理论课题(MMJJ20180202)。

作者简介：杨 腾(1996-)，男，硕士研究生，主要研究方向：软件工程、代码搜索；赵逢禹(1963-)，男，博士，教授，CCF 会员，主要研究方向：计算机软件与软件系统安全、软件工程与软件质量控制、软件可靠性；刘 亚(1983-)，女，博士，副教授，CCF 会员，主要研究方向：信息安全、密码学。

收稿日期：2021-03-24

近年来，研究人员开始将机器学习和深度学习技术应用于代码搜索领域。Gu 等人[6]提出一个基于 LSTM 的代码搜索模型 DeepCS，该模型利用了代码的注释信息，通过深度神经网络得到代码和注释的向量表示，模型训练时通过相似度计算，使得匹配的代码和注释生成相似的向量。与基于信息检索的代码搜索技术相比，运用深度神经网络通过对比代码和查询语句的向量之间的空间距离能够挖掘到更高层次的特征，更充分地理解代码语义。

利用深度学习进行代码搜索，通常需要先从源代码中提取代码特征，以便更充分地理解代码语义。比如 Aroma[7]从简化的抽象语法树中提取了下文关系作为代码特征、DeepCS 通过遍历抽象语法树提取了由代码调用的 API 组成的序列作为代码特征。本文认为基于抽象语法树的特征提取方法提取的主要是代码的结构特征，没有考虑到代码中变量的定义、判断、使用的数据流信息，不能够很好地反映出代码的语义特征。而程序切片通过分析程序语句之间的控制依赖和数据依赖关系，移除了程序中与兴趣点无关的部分，能够更好地反映出源代码的结构和功能信息。

基于这一思想，本文构建了基于程序切片和 BiGRU[8]的代码搜索模型。首先从开源代码库获取数据集，提取出方法代码-功能描述对。建立方法代码的程序依赖图，将出度最大的节点作为兴趣点，在程序依赖图上利用图形可达性算法获得程序切片。然后使用 BiGRU 神经网络模型，结合注意力机制，构建一个注释-代码联合嵌入模型，将方法代码和功能描述当作输入，训练模型学习二者的相关性。在搜索阶段，用户将需要的功能描述作为输入，该模型会返回与输入信息匹配的方法代码。

1　基于程序切片和 BiGRU 的代码搜索

为了更加准确高效地搜索具有某些功能的代码，本文提出了一个基于程序切片和 BiGRU 的代码搜索方法。该方法首先从开源代码库 GitHub 上获取数据集，从中选取部分具有清晰完善注释的方法代码，构建出方法代码-注释对。然后为方法代码建立程序依赖图，在程序依赖图上利用可达性算法获得程序切片，从程序切片中提取代码的结构特征。最后将提取到的特征和相应的注释输入进 BiGRU 网络模型，通过相似度计算训练模型。

本文构建了一个基于程序切片和 BiGRU 的代码搜索模型。其中，程序切片是通过在程序依赖图上做可达性分析得到的，具体包含了对源代码的控制依赖信息和数据依赖信息的分析，能够更加准确地表达代码语义。由于在数据量不是特别大的时候，GRU 能够达到和 LSTM[9]相似的性能，但参数量比 LSTM 减少了三分之一，收敛速度更快，因此本文选择使用 BiGRU 神经网络模型。此外，该模型还增加了注意力机制，通过加权处理，突出重点，使模型表现出更好的性能。与 DeepCS 相比，在性能相当的前提下，本文提出的模型收敛速度更快。代码搜索模型的构建流程如图 1 所示。由图 1 可知，对其中主要部分的功能解析可做阐释分述如下。

(1)功能描述提取。首先从 Github 上下载 Java 项目，从中提取方法和注释对构建代码库，然后对方法注释进行解析，只保留其中对方法的功能的描述。

(2)方法特征提取。由图 1 可知，在特征提取阶段，该模型生成了源代码的程序依赖图 PDG，从中选取出度最大的语句作为兴趣点，利用可达性算法构建程序切片。此外，该模型还生成了抽象语法树 AST，从中提取了方法名、返回值类型、参数等特征，这些特征也与方法的功能有着密切的关系，将其与程序切片一同作为方法特征。通过完全连接层将这 4 个特征融合到一个向量中，即：

$$\boldsymbol{C} = \tanh(\boldsymbol{W}^{c}[\boldsymbol{m};\boldsymbol{r};\boldsymbol{p};\boldsymbol{s}]) \tag{1}$$

其中，向量 $\boldsymbol{C}$ 为融合后的向量；$\boldsymbol{W}^{c}$ 是可训练参数的矩阵；$[\boldsymbol{m};\boldsymbol{r};\boldsymbol{p};\boldsymbol{s}]$ 是方法名、返回值类型、方法参数和程序切片四个特征对应向量的串联。

(3)模型训练。在模型训练阶段，将上个阶段提取的方法代码特征和功能描述嵌入到高维向量空间，得到两者各自的向量表示。通过训练学习二者的相关性，如果功能描述和方法代码具有相似的语义，那么对应的向量表示就应该互相接近。

在查询阶段，由用户输入一句对代码功能的自然语言描述，模型把该描述嵌入到向量空间中，在代码库中搜索与其相似度最高的 5 个方法的向量表示，然后从源代码库中取出具有相应功能的方法代码作为结果返回。

2　程序切片和代码搜索模型关键技术

本文用到的关键技术主要有获取源代码的程序切片、BiGRU 和 Attention 机制。这里拟对此展开研究论述如下。

2.1　程序切片

程序切片的概念由 Weiser[10]首次提出，研究发现某个输出只与部分语句相关，将不相关的语句删

除不影响该输出的结果,将这种只与某个输出相关的语句构成的程序称为程序切片。

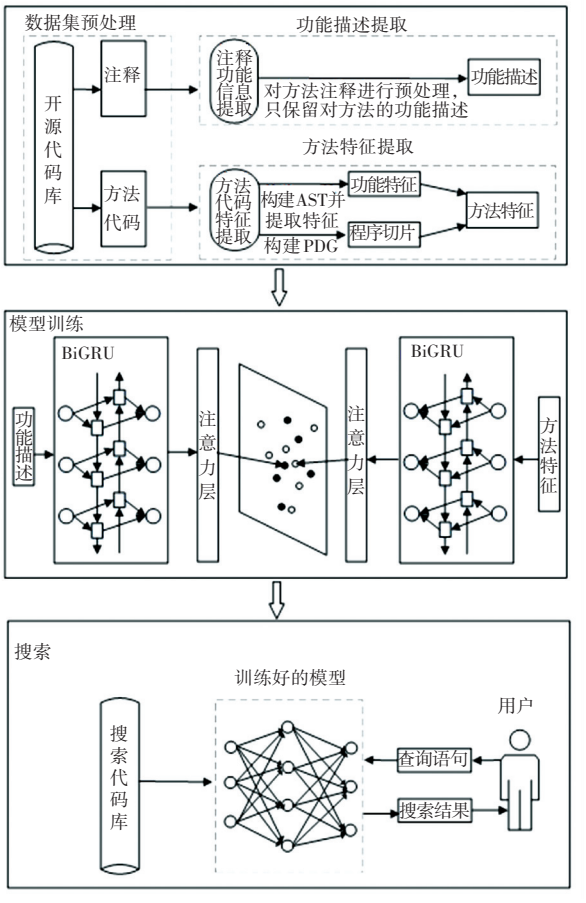

图 1 代码搜索模型构建流程

Fig. 1 Code search model construction process

目前研究程序切片的主要方法是基于程序依赖图来做的,Ottenstein 等人[11]引入了程序依赖图的图可达性算法,用来计算程序切片。李润清等人[12]通过提取源代码的切片摘要,把切片摘要作为搜索引擎中的关键字进行搜索,证明了运用程序切片的代码搜索引擎的性能要优于传统的代码搜索方法。本文以 Java 代码为例,给出了构建程序切片的方法。

要获取程序切片,需要为 Java 方法生成相应的程序依赖图 PDG(Program Dependence Graph),PDG 中的节点表示语句或者谓词表达式,边表示节点之间的数据依赖、控制依赖关系。控制依赖指的是因控制流引起的模块之间的依赖关系。数据依赖指的是由数据流引起的变量之间的依赖关系。PDG 图是一个有向图,本文选择程序依赖图中除入口节点以外出度最大的节点对应的语句 N 作为兴趣点,因为出度代表受其影响的语句数量,出度越大则说明受到该节点影响的语句数量越多,意味着该节点对应的语句蕴含的结构信息越丰富。因此选择这样的语句作为兴趣点 N,递归遍历数据依赖或控制依赖于 N 的结点、即 PDG 中以 N 为源节点的邻接节点,将其加入到切片集合,生成节点 N 的前向切片。

对于 PDG 中的每个节点,文中使用集合 $S.out$ 代表以节点 S 为源节点的邻接节点的集合。算法 1 给出了获取 PDG 中出度最大的结点 max 的过程,算法 2 描述了以 max 为兴趣点构建程序切片的步骤。算法流程详见如下。

算法 1 *findMax*

输入:方法代码 Method

输出:出度最大的节点 max

初始化: $Queue, max, visited = \Phi, p \in PDG$

$PDG = ConstructPDG(Method)$

$p = PDG.getNode(0)$ // p 是 PDG 中入度为 0 的结点

$max = p$;

$Queue.enterQueue(p)$ // p 入队

While (! $Queue.empty()$)

  $tmpNode = Queue.DeQueue()$

  If ($tmpNode \notin visited$)

    $visited.add(tmpNode)$

    If $max.outDegree < tmpNode.outDegree$

      $max = tmpNode$

    End If

  End If

  For $node$ in $tmpNode.out()$

    If $node \notin visited$

      $Queue.enterQueue(node)$

    End If

  End For

End While

return max

算法 2 *constructSlice*

输入:程序依赖图 PDG,节点 max

输出:程序切片 $Slice = \{n_1, n_2, \cdots, n_k\}$

初始化: $Slice = \Phi, visited = \Phi$

$Slice.add(max)$ // 将 max 加入到 $Slice$

If $max \notin visited$ // 遍历节点 max 的输出节点集合

$visited.add(max)$ // 标记 max 被访问过

For $node$ *in* $max.out()$

  $constructSlice(node)$

End For

End If

Return Slice

2.2　双向门控循环单元

双向门控循环单元(GRU)[13]是长短期记忆 LSTM 模型的一个变体，只有 2 个门函数：更新门和重置门。其中，更新门用于控制前一时刻的状态信息被带入到当前状态中的程度，更新门的值越大说明前一时刻的状态信息带入越多。重置门控制前一状态有多少信息被写入到当前的候选集上，重置门越小，前一状态的信息被写入得越少。

GRU 模型结构图如图 2 所示。图 2 中，r_t 为控制重置的门控(reset gate)，z_t 为控制更新的门控(update gate)，h_{t-1} 为前一时刻的状态信息，h_t 为当前时刻的状态信息，σ 为 sigmoid 函数，通过这个函数可以将数据变换为 0~1 范围内的数值，从而来充当门控信号。具体计算如下：

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{2}$$

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{3}$$

$$\tilde{h}_t = \tanh(W_{\tilde{h}_t} \cdot [h_{t-1}, x_t]) \tag{4}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{5}$$

其中，W_r、W_z、$W_{\tilde{h}_t}$ 是通过训练学习的参数，$\tilde{h}_t$ 是候选隐藏层状态，并只与输入 x_t、上一层的隐藏状态 h_{t-1} 有关。

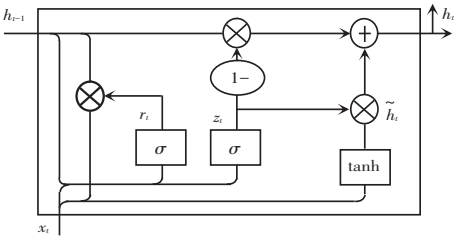

图 2　GRU 结构图

Fig. 2　GRU structure diagram

BiGRU 由 2 个方向相反的 GRU 构成。一个正向的 GRU，利用历史信息；一个反向的 GRU，利用未来的信息。这样在时刻 t，既能够利用到 $t-1$ 时刻的信息，又能利用 $t+1$ 时刻的信息。在每一时刻，把输入同时给 2 个 GRU，输出由 2 个 GRU 的状态同时决定。BiGRU 的结构如图 3 所示。

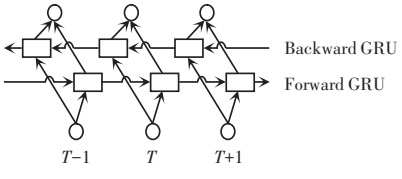

图 3　BiGRU 结构

Fig. 3　BiGRU structure diagram

2.3　Attention 机制

注意力机制(attention mechanism)源自人类大脑对新事物认知的特点，即对于重要的内容分配较多注意力，而对于不重要的部分分配较少的注意力。借助注意力机制为神经网络隐层单元分配不同的概率权重，能够关注到更有利的特征，同时降低对一些冗余信息的关注[14]。Attention 通过对数据自动加权变换，把 2 个不同的部分联系起来，突出重点，以表现出更好的性能。引入 Attention 机制的重要原因就是减少文本序列中关键信息的丢失。由于注意力机制在自然语言处理中对强化关键信息的有效性，所以本文在双向 GRU 神经网络的基础上添加了 Attention 机制来提高模型的准确率。Attention 机制注意力分布的计算通常分 2 步：

(1)给定一个查询向量 $\boldsymbol{q}$，通过其与输入向量 $[X_1, X_2 \cdots\cdots X_n]$ 的相关性，使用打分函数计算得到 n 个 *score*，相关性越高，则值越大。

(2)通过 *softmax* 函数计算得到注意力分布，计算方法如公式(6)所示：

$$\alpha_i = softmax(s(key_i, \boldsymbol{q})) \tag{6}$$

注意力分布 α_i 可以解释为在上下文查询 $\boldsymbol{q}$ 时，第 i 个信息受关注的程度。$s(x_i, \boldsymbol{q})$ 为注意力打分函数，计算方法如公式(7)所示：

$$s(x_i, \boldsymbol{q}) = \frac{\boldsymbol{x}_i^{\mathrm{T}}}{\sqrt{d}} \tag{7}$$

其中，$\boldsymbol{x}_i^{\mathrm{T}}$ 表示输入向量；$\boldsymbol{q}$ 表示查询向量；d 表示输入向量的维度。缩放点积模型解决了点积模型在 d 较大的时候的结果方差会比较大，*softmax* 的梯度会变小的问题。

3　实验与分析

本节给出了该模型的实现步骤，并通过搜索结果的准确率和检索排名等指标来检验该模型的性能。

3.1　实验数据集

本文通过从开源网站 Github 上下载 Java 项目来构建训练代码库和搜索代码库，从中选择了 268 个 Java 项目。在这些 Java 代码中选取了11 893个方法-注释对作为正样本，然后构建了10 224个负样本一同作为训练代码库。正样本就是数据集里匹配的方法-注释对，负样本则是随机选取的注释与当前方法构成的方法-注释对。搜索代码库是由数据集中的所有方法代码构成的，其中包括没有注释信息的方法，共有35 533个方法代码。数据集 80%作

为训练集,20%作为测试集,使用 BiGRU 模型在训练集上进行训练并在验证集上对模型进行调参,最后在测试集上测试。

3.2 参数设置

本文将词向量维度设置为 100,BiGRU 每个方向上的隐藏层节点数设置为 50, *batch_size* 设置为 64, *dropout_rate* 为 0.5,以防止模型发生过拟合现象。

3.3 实验过程

(1) 在对注释信息进行抽取时,去除@ author、@ see、@ link、@ since 等注解后面的信息,只保留注释中对方法的功能描述。本文从 Github 获取了停用词列表,然后使用 NLTK 分词对功能描述进行分词,根据停用词列表去除分词后的文本中的停用词。

(2)取出数据集中的方法代码,通过 2.1 节的算法 1 和算法 2 构建方法代码的程序切片。

(3)使用 Eclipse 的 JDT 工具生成方法代码的抽象语法树,遍历抽象语法树提取返回值类型、方法名、参数类型,将其与程序切片一同作为方法特征。

(4)分别按顺序存储方法特征和功能描述,通过按顺序分别读取方法特征和功能描述的方式输入数据,使用 word2vec[15] 获得方法代码特征和功能描述各自的词向量表达,使用 H5 格式的文件存储这些数据。

(5)本文使用 torch.nn.gru 构建了 BiGRU 模型,将方法特征和功能描述的词向量表达作为 BiGRU 的输入,提取深层局部特征和长距离特征。将 BiGRU 提取的特征作为 Attention 层的输入,并在 Attention 分配词向量权重。在完全连接层使用 Keras 的 *Concatenate* 函数对方法代码的 4 个特征进行融合,得到嵌入向量。训练时使正样本中的功能描述在向量空间中不断地向方法代码靠近,负样本中的功能描述在向量空间中不断地远离方法代码。在训练到 340 个迭代周期左右, 发现 *loss* 不再继续减小,停止训练。

3.4 实验结果与分析

本小节将在搜索代码库上对本文提出的方法进行评估,为了评估提出的代码搜索方法的有效性,本文从准确率和检索排名等方面将该方法与其他方法作对比。

本文选择了 *Precision* 和 *MRR* (Mean reciprocal rank)对搜索结果进行评价,这 2 个指标都是信息检索领域常用的评价指标。总地说来, *Precision* 计算的是正确预测为正样本的数量占所有预测为正样本的数量的比例。*MRR* 通过正确检索结果在全部检索结果中的排名来评估检索系统的性能。

本文在同一实验环境下,使用相同的数据集将本文模型与 Sourcerer、SWIM 以及 DeepCS 做对比,用来评估该模型的性能。Sourcerer 是基于 Lucene 的代码搜索工具,能够代表传统的基于全文检索技术的搜索引擎。SWIM 利用词袋技术将查询翻译成 API,考虑到了代码的结构信息。对比结果见表 1。

表 1 实验结果对比

Tab. 1 Comparison of experimental results

Model	*Precision*/ %	*MRR*
Sourcerer	41.5	0.45
SWIM	45.6	0.48
DeepCS	47.4	0.67
Our Model	50.6	0.72

为了验证本文提取特征的有效性以及模型对结果的影响,增加了特征提取方式与模型组合的对照实验。实验以方法名、返回值类型以及方法参数等特征结合 LSTM 的模型作为基线模型,在此基础上分别验证了使用了程序切片作为代码特征的效果以及使用 BiGRU 替换 LSTM 的效果。模型对比效果见表 2。

表 2 模型效果对比

Tab. 2 Comparison of model effect

Model	*Precision*/ %	*MRR*
LSTM	43.5	0.41
BiGRU	42.8	0.42
LSTM + Program Slice	49.8	0.53
BiGRU + Program Slice	50.6	0.55

由表 1 可知, 对于 *Precision*, 本文模型的表现与 DeepCS 相当,比 Sourcerer 和 SWIM 要好。*MRR* 的值也表明,本文模型正确的查询结果更加靠前。由表 2 可知,提取程序切片作为代码特征提高了搜索结果的精度,BiGRU + Program Slice 的模型达到了与 LSTM + Program Slice 模型相当的性能,但在实验过程中发现该模型的收敛速度比 LSTM + Program Slice 模型快。LSTM + Program Slice 在 420 个迭代周期后 *loss* 不再发生明显下降,BiGRU + Program Slice 则在 340 个迭代周期以后 *loss* 不再下降。虽然本文模型在整体性能上比 SWIM 和 Sourcerer 表现得好,但是研究发现在个别查询语句的结果中,Sourcerer 的准确率超过了本文模型。在分析后,研究发现这是因为这些查询语句中包含了较多与代码中变量名相同的关键词,在这种情况下,基于全文检索技术的方法的优势就体现了出来。这说明本文的模型还有需要继续改进的地方,在后续工作中会改进这些问题。

4　结束语

本文提出了基于程序切片和 BiGRU 的代码搜索模型，该模型通过构建代码的程序切片与代码功能特征一同作为方法特征，然后计算方法特征和注释的向量表示的余弦相似度，使得匹配的功能描述和方法生成相似度较高的向量，在查询时将在向量空间中与查询语句的向量表示最接近的方法返回给用户。实验结果表明，跟传统的代码搜索方法相比，该模型通过构建程序切片，有效地利用了代码的控制依赖和数据依赖关系，更加准确地提取了代码的语义信息，在一定程度上提高了搜索结果的准确率。下一步研究工作将是对构建程序切片的方法进行改进，比如选择多个兴趣点构建多个切片，以挖掘更准确的代码语义信息。

参考文献

[1] 黎宣，王千祥，金芝. 基于增强描述的代码搜索方法[J]. 软件学报，2017，28(6)：1405-1417.

[2] SINDHGATTA R. Using an information retrieval system to retrieve source code samples[C]// International Conference on Software Engineering. Shanghai，China：dblp，2006：905-908.

[3] HILL E，POLLOCK L L，VIJAY-SHANKER K. Improving source code search with natural language phrasal representations of method signatures [C]// IEEE/ACM International Conference on Automated Software Engineering. Lawrence，KS，USA：ACM，2011：524-527.

[4] BAJRACHARYA S K，OSSHER J，LOPES C V. Leveraging usage similarity for effective retrieval of examples in code repositories[C]//ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York，NY，USA：ACM，2010：157-166.

[5] RAGHOTHAMAN M，WEI Y，HAMADI Y. SWIM：Synthesizing what I mean - code search and idiomatic Snippet synthesis [C]// 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). Austin，TX：IEEE，2016：357-367.

[6] GU X，ZHANG H，KIM S. Deep code search [C]// IEEE/ACM 40th International Conference on Software Engineering (ICSE). Gothenburg：IEEE，2018：933-944.

[7] LUAN Sifei，YANG Di，BARNABY C，et al. Aroma：code recommendation via structural code search[C]//Proceedings of the ACM on Programming Languages. Athens：ACM，2019：1-28.

[8] ZHANG Liujie，ZHOU Yanquan，DUAN Xiuyu，et al. A hierarchical multi-input and output Bi-GRU model for sentiment analysis on customer reviews[J]. Iop Conference，2018，322(6)：062007.

[9] 刘丹，叶茂. 回复式神经网络及其应用研究综述[J]. 小型微型计算机系统，2020，41(10)：2024-2029.

[10] WEISER M . Program slicing[J]. The IEEE Transactions on Software Engineering，1984，10(4)：352-357.

[11] OTTENSTEIN K J，OTTENSTEIN L M.The program dependence graph in a software development environment[J]. ACM Sigplan Notices，1984，19(5)：177-184.

[12] 李润青，曾国荪. 程序源代码中的切片摘要提取及在搜索中的应用[J]. 信息技术与网络安全，2018，37(3)：122-125，130.

[13] DEY R，SALEMT F M. Gate-variants of Gated Recurrent Unit (GRU) neural networks [C]// IEEE International Midwest Symposium on Circuits & Systems. Boston，MA，USA：IEEE，2017：1597-1600.

[14] 司念文，王衡军，李伟，等. 基于注意力长短时记忆网络的中文词性标注模型[J]. 计算机科学，2018，45(4)：66-70，82.

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

(上接第 215 页)

[3] SRIDEVI K，SUNDARAMBAL M，DHARAN K M，et al. Hand gesture recognition system using radial basis function Neural Networks [J]. Journal of Innovation in Electronics and Communication Engineering，2017，7(2)：38-41.

[4] 李逸琳，陶静，霍艺文，等. 手势特征提取与图像分割的优化研究[J]. 计算机应用与软件，2020，37(2)：161-165，206.

[5] 程冉，史健芳. 基于卷积神经网络的手势识别算法研究[J]. 电子设计工程，2021，29(2)：179-184.

[6] LI Yalan，LU Ruhua，HUANG Rui，et al. Research on face recognition algorithm based on HOG feature [J]. Journal of Physics：Conference Series，2021，1757(1)：012076.

[7] JUNAIDY D，WULANDARI M，TANUDJAJA H. Real time face detection using haar-like feature method and local binary pattern method [J]. IOP Conference Series：Materials Science and Engineering，2019，508(1)：012099.

[8] EMADI M，EMADI M. Human face detection in color images using fusion of Ada Boost and LBP feature[J]. Majlesi Journal of Telecommunication Devices，2020，9(1).

[9] STERGIOPOULOU E，SGOUROPOULOS K，NIKOLAOU N，et al. Real time hand detection in a complex background [J]. Engineering Applications of Artificial Intelligence，2014，35：54-70.

[10] 翁汉良，战荫伟. 基于视觉的多特征手势识别[J]. 计算机工程与科学，2012，34(2)：123-127.

[11] 杨学文，冯志全，黄忠柱，等. 结合手势主方向和类-Hausdorff 距离的手势识别[J]. 计算机辅助设计与图形学学报，2016，28(1)：75-81.

[12] 秦育罗，郭冰，孙小荣. 改进 Hausdorff 距离及其在多尺度道路网匹配中的应用[J]. 测绘科学技术学报，2020，37(3)：313-318.

[13] 肖宇. 基于序列图像的手势检测与识别算法研究[D]. 成都：电子科技大学，2014.

[14] 卢梦圆，官巍，马力. 基于多特征融合的手势识别研究[J]. 计算机与数字工程，2020，48(9)：2157-2161.

[15] 赵倩楠，胡延平. 一种基于特征融合的手势识别方法[J]. 物联网技术，2020，10(9)：33-36.

[16] ANDREW A M. An Introduction to Support Vector Machines and other kernel-based learning methods[J]. Robotica，2000，18(6)：687-689.

[17] 刘斌，米强，徐岩. LBP 和 MB-LBP 加权融合的人脸识别[J]. 计算机工程与设计，2018，39(2)：551-556.

[18] 黄果，许黎，蒲亦非. 分数阶微积分在图像处理中的研究综述[J]. 计算机应用研究，2012，29(2)：414-420，426.