龙腾,刘建勋,张祥平,等.基于指针网络的双输入代码补全[J].湖南科技大学学报(自然科学版),2025,40(2):59-70.doi:10.13582/j.cnki.1672-9102.2025.02.008

LONG T, LIU J X, ZHANG X P, et al. Dual-Input Code Completion Based on Pointer Networks[J]. Journal of Hunan University of Science and Technology (Natural Science Edition),2025,40(2):59-70.doi:10.13582/j.cnki.1672-9102.2025.02.008

基于指针网络的双输入代码补全

龙腾[1],刘建勋[2,3*],张祥平[2],扈海泽[2],曹奔[2],成思强[2]

(1.湖南涉外经济学院 信息与机电工程学院,湖南 长沙 410205;2.湖南科技大学 计算机科学与工程学院,湖南 湘潭 411201;
3.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)

摘　要:现有的代码补全方法依赖于抽象语法树(Abstract Syntax Tree, AST),不能完全捕获源代码上下文的语法结构信息和语义信息,导致代码补全的准确率较低.文章提出一种基于指针网络的双输入代码补全方法(Dual-Input Code Completion Based on Pointer Network, DIBPN).DIBPN 将源代码的 AST 序列和 Token(标识符)序列共同作为模型输入,再通过指针混合网络预测下一个 Token,同时采用深度学习对特征信息进行提取和表征,最后,对 DIBPN 模型进行试验验证.结果表明:与 3 种基准模型相比,DIBPN 具有更优的试验结果.因此,DIBPN 能提高代码补全的准确率,进而提高软件开发的效率.

关键词:代码补全;AST;指针网络;深度学习

中图分类号:TP311.5　　**文献标志码**:A　　**文章编号**:1672-9102(2025)02-0059-12

Dual-Input Code Completion Based on Pointer Networks

LONG Teng[1], LIU Jianxun[2,3], ZHANG Xiangping[2], HU Haize[2], CAO Ben[2], CHENG Siqiang[2]

(1. College of Mechatronics and Information Engineering, Hunan International Economics University, Changsha 410205, China;
2. School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China;
3. Hunan Provincial Key Laboratory For Services Computing and Novel Software Technology, Hunan University of Science and Technology, Xiangtan 411201, China)

Abstract: The existing code completion methods mainly rely on Abstract Syntax Tree (AST). However, AST parsing-based methods cannot fully capture the source code contextual syntax and semantic information, resulting in low accuracy of code completion. Thus, we propose a dual-input code completion method based on Pointer Networks (DIBPN). DIBPN takes the AST sequence of source code and Token sequence together as the model input, then predicts the next Token by pointer hybrid network, and uses deep learning to extract and characterize the feature information. The experimental results show that DIBPN has better experimental results relative to the three benchmark models. Therefore, DIBPN proposed in the paper can improve the accuracy of code completion, thus improving the efficiency of software development.

Keywords: code complementation; Abstract Syntax Tree (AST); pointer networks; deep learning

随着信息化时代的发展,软件产品的需求越来越复杂,提高软件的开发质量和效率必然成为软件工程研究者们关注的一个核心问题.代码补全技术被认为是提高软件开发自动化水平的重要方法,是现代集成开发环境(Integrated Development Environment, IDE)的重要组成部分,能够有效地帮助开发人员预测代码

收稿日期:2022-11-15

基金项目:国家自然科学基金资助项目(61872139)

* **通信作者**,E-mail:1797880502@qq.com

的类名、方法名和关键字等,减少编码过程中的拼写错误,从而提高软件的开发效率.

现有的代码补全方法主要分为 2 种:一种方法利用静态类型信息结合各种启发式规则决定要预测的标识符(Token),如变量名和方法名等[1-10],这种方法很少考虑代码上下文的语义信息,如 Eclipse 中的代码补全插件 CACHECA 利用静态类型信息为用户推荐补全内容,其推荐结果按照字母顺序进行排列展示[11-15];另一种方法利用已有的代码样例和前文语义来预测 Token[16-24].随着机器学习和深度学习的发展,越来越多的研究者们将深度学习方法应用于代码补全任务中[25-34].深度学习方法可以从大规模代码中学习代码 Token 之间的概率分布,从而提高 Token 推荐的准确率.HINDLE 等[35]首次提出对代码 Token 序列进行概率建模;TU 等[36]在 HINDLE 等[35]的基础上提出程序中的 Token 在局部范围内具有一定的重复性,从而引入缓存机制来维护局部特征,使补全效果得到较大提升;FRANKS 等[11]开发了名为 CACHECA 的 Eclipse 插件,证实局部重复规律对于代码补全的有效性;VINYALS 等[37]利用程序语言具有局部重复的规律,将指针网络机制加入语言模型中,可以准确预测局部重复出现的词;BHOOPCHAND 等[38]提出一种精简指针网络,主要解决自定义标识符预测的问题(如类名、方法名和变量名等);HELLENDOORN 等[39]通过对比循环神经网络与带有缓存机制的 N-Gram,发现代码的局部性特征对于 Token 的预测具有极大帮助;LI 等[40]提出一种结合注意力机制和指针网络的进行代码补全的指针混合网络模型,该模型对抽象语法树(Abstract Syntax Tree,AST)序列进行建模,将其作为模型输入,通过 2 个组件和 1 个选择器来决定是利用全局 RNN 组件在全局词表中预测下一个词,还是利用指针网络从输入的代码片段中复制一个词.该模型在一定程度上提升了代码补全的准确率,但是模型的输入只有 AST 序列,AST 是对源代码进行抽象概括,原始代码片段中的许多细节符号并没有完全在 AST 中体现出来,如标点符号和运算符等细节类 Token,这使得程序中的语义信息不完整,进而影响代码补全的准确性.

针对以上问题,文章提出一种基于指针网络的双输入代码补全模型(Dual-Input Code Completion Model Based on Pointer Network, DIBPN).DIBPN 将 AST 序列与 Token 序列共同作为模型输入,以便同时提取代码的语法结构信息和语义信息,再利用指针混合网络从全局词表或者局部上下文中预测下一个 Token.

文章在 JavaScript(JS)和 Python(PY)数据集(http://plml.ethz.ch)上进行试验评估,在指针混合网络的基础上增加 Token 序列作为输入,同时加入残差连接[41-44].通过增加输入,弥补指针混合网络忽略的语义信息,加入残差连接以保证源代码信息的完整性.

为了评估所提模型 DIBPN 的有效性,采用 LSTM 模型,Attentional LSTM 模型和 Pointer Mixture Network 模型作为基准模型进行对比分析.在对比分析的过程中,采用评价指标 Accuracy 对以上模型进行评估分析.

文章的主要贡献如下:

1)提出基于指针网络的双输入代码补全模型(DIBPN),该模型中的双输入结构利用 AST 序列和 Token 序列分别提取源代码的语法结构信息和语义信息.DIBPN 从语法和语义 2 个角度联合学习,可以更好地预测细节类 Token.

2)在 JS 和 PY 这 2 个数据集上进行试验分析,分别对比 3 种基准模型在代码补全上的有效性.试验结果表明本文所提出的 DIBPN 模型对 Token 补全更有效.

文章的结构如下:第 1 部分讨论代码补全所涉及的背景基础,主要包括 AST 表征、Token 表征和指针混合网络模型;第 2 部分从预处理、训练和补全 3 个阶段详细介绍 DIBPN 模型;第 3 部分介绍文章使用的试验数据集并详细讨论所研究的问题,根据研究问题对模型试验进行介绍,并对试验结果进行分析;第 4 部分对文章中的研究内容以及贡献进行总结,并对未来所需要做的工作进行展望.

1 研究背景

代码补全研究相关的背景知识主要包括:(1)Token 表征,阐述如何将 Token 转成向量;(2)指针混合网络,阐述如何通过指针网络进行代码补全;(3)问题定义,表明文章的研究动机.

1.1 Token 表征

现有的代码补全方法中,大部分方法都将源代码转换成 AST 的形式作为模型输入,Python 代码段及

其对应的抽象语法树如图 1 所示.AST 能够提取源代码的语法结构信息,但无法表示源代码的一些细节以及语义信息,如“in”“(”“)”“:”等.因此,考虑将源代码进行 Token 表征后作为模型输入,以充分利用代码的语义信息.在构建词汇表时,由于源代码的词汇量非常大,会影响模型的训练速度,因此,只选择训练集中使用频率最高的 K 个单词构建 Token 词汇表.

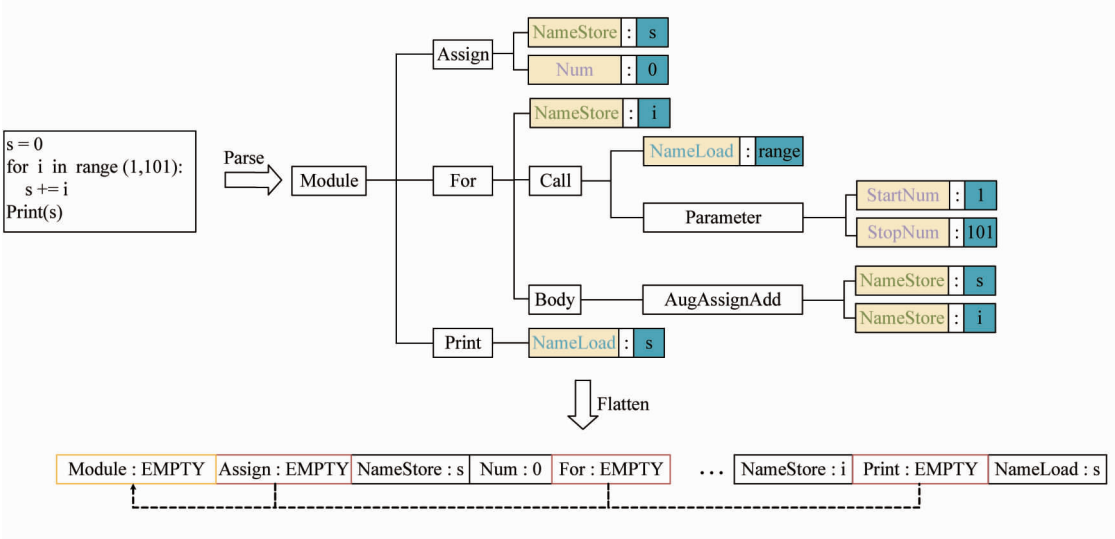

图 1　Python 代码段及其对应的抽象语法树

1.2　指针混合网络

在代码补全中,通常选择使用频率最高的 K 个单词构建全局词表,未在词表中的词称为未知词(Out-of-Vocabulary, OOV),代码补全是在全局词表中进行预测,因此它们无法被补全,这种现象称为 OOV 问题.LI 等[40]提出 Pointer Mixture Network 模型,利用程序语言的局部重复性特征进行代码补全,并且能在一定程度上解决 OOV 问题.Pointer Mixture Network 模型包含 2 个组件,全局 RNN 组件从预定义的全局词表中预测下一个 Token,局部指针组件从局部上下文中复制 1 个词作为下一个 Token.用户自定的变量名、方法名和类名等在局部上下文中的使用频率相对较高,根据代码的局部重复性特征,Pointer Mixture Network 模型可以通过局部指针组件进行预测,能够有效缓解 OOV 问题.Pointer Mixture Network 模型结构如图 2 所示[40].

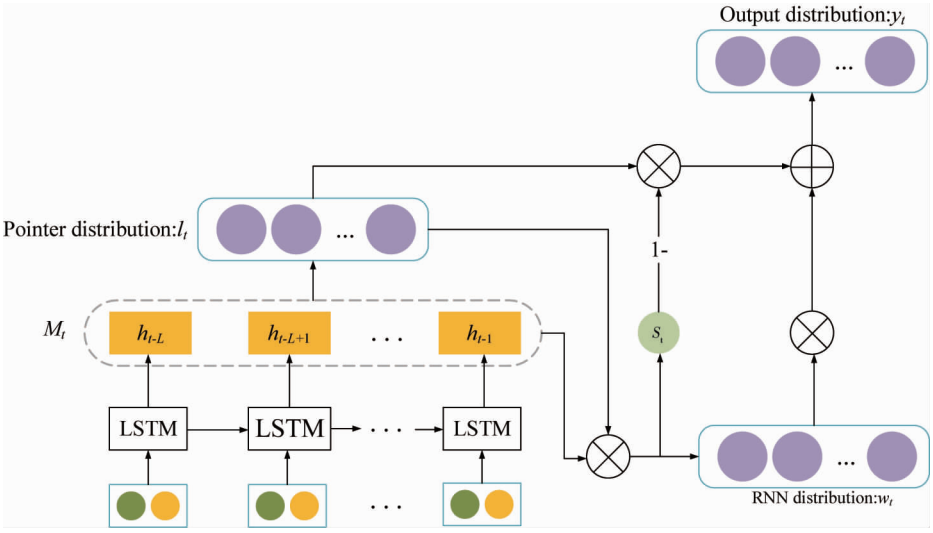

图 2　Pointer Mixture Network 模型结构

1.3　问题定义

所提出的 DIBPN 模型旨在解决细节类 Token 无法补全以及语义信息缺失的问题.Pointer Mixture

Network 模型以节点向量为输入,充分利用源代码的语法结构,但 AST 并不会表示出真实语法的每一个细节,因此,细节类 Token 无法补全.这类 Token 对于补全并非毫无意义,若能对这部分 Token 进行补全,将极大地提高开发效率.同时,只以节点向量为输入,虽然能充分利用源代码的语法结构信息,但忽略了其语义信息.源代码的语义信息有助于对代码的理解,若对其加以利用,也能够提高代码补全的准确率.

2 DIBPN 的提出

所提出的 DIBPN 模型是基于指针网络的双输入代码补全模型,代码补全流程如图 3 所示,主要分为 3 个阶段:(1)预处理阶段;(2)训练阶段;(3)补全阶段.

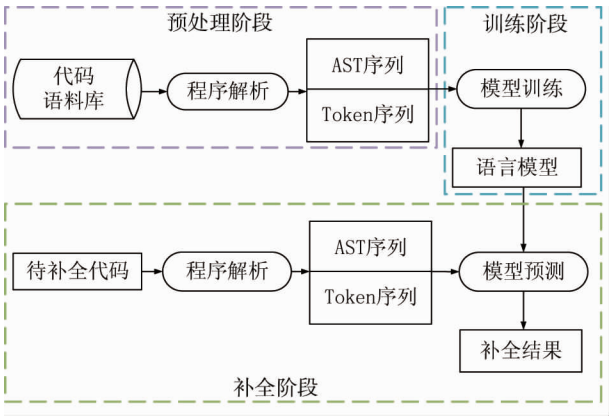

图 3 代码补全流程

2.1 预处理阶段

为了使神经网络模型更好地学习代码特征,需要对代码数据进行预处理操作,对源代码进行不同方式的预处理会直接影响后续的模型构建.由于文章采用的是双输入,所以源代码片段及其对应的 AST 都要进行预处理.本节主要介绍经过代码预处理得到的主要表现形式,分别为 Token 序列和 AST 序列.

2.1.1 Token 序列

在自然语言处理中,对于文本中的输入数据,常用的预处理方法是将文本中的段落分为若干句,并且将每一句分为若干词,结合构建的词典,使模型识别输入的文本.同理,代码可以被视为一种特殊的文本数据.通过使用文本的预处理方式对代码进行预处理,首先,将源代码通过词法分析消除代码中无意义的字符,如空格、换行符或者注释等,并将代码分为若干词,每个词都称为 Token,再将它们 Token 化为与源代码等价的 Token 序列.Python 语言的程序语句可以转化为 Token 序列的形式,包括 Python 内置的关键字(for, if 等)、数字(整数、小数)、操作符({, *, %, & 等)和自定义标识符(变量名、类名等).例如,对于语句‘for i in range(1,101):’,经过分词、Token 化后,可以转化为 Token 序列[for, i, in, range, (, 1, 101,),:].

2.1.2 AST 序列

AST 是以树结构形式表示源代码的最常用的方法之一,可以充分体现出源代码的结构信息.将解析源代码得到的 AST 以深度优先搜索的方式转换成 AST 序列,并增设 2 位表示 AST 节点是否有孩子节点或右兄弟节点,以此获得 AST 节点之间的父子关系.

2.2 训练阶段

基于指针网络的双输入代码补全模型的结构如图 4 所示,模型主要分为输入层、编码层、注意力层和输出层 4 个部分.

2.2.1 输入层

输入层主要处理模型的输入,所提出的双输入结构包含 2 个向量,分别是节点向量和 Token 向量,该结构能够充分提取源代码的语法结构信息和语义信息.

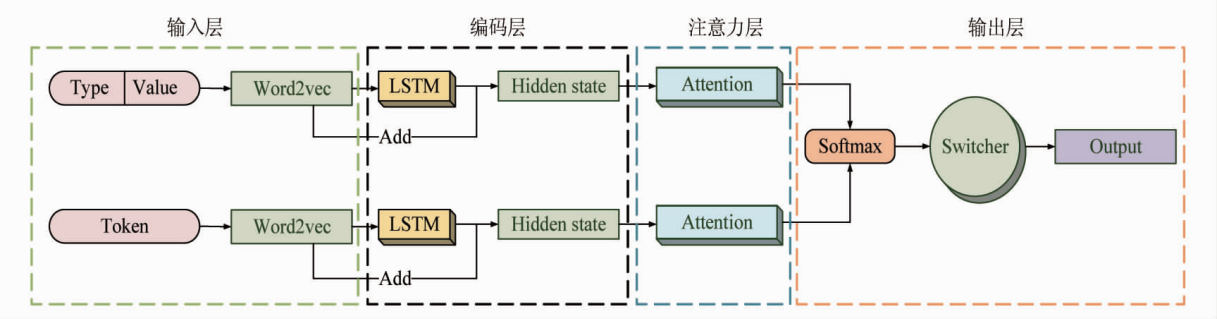

图 4　基于指针网络的双输入代码补全模型的结构

AST 序列通过树结构的形式表示源代码,能充分体现源代码的语法结构信息如父子关系等.AST 节点包含节点类型 Type 和节点值 Value 这 2 个部分,二者之间以“:”隔开.在 AST 中,只有终端节点的 Value 表示代码中的 Token,这些 Token 构成代码的主要结构,但并不能表示代码中的细节类 Token.对于非终端节点,其 Type 体现了代码的语法结构且没有 Value 值,为保持节点的结构一致,文章用特殊字符“EMPTY”代替.本文定义 $\boldsymbol{\omega}_i=(\boldsymbol{T}_i,\boldsymbol{V}_i)$ 为 1 个 AST 节点,那么每个程序都可以表示为单词序列 $\omega=\{\boldsymbol{\omega}_1,\boldsymbol{\omega}_2,\cdots,\boldsymbol{\omega}_m\}$,其中,$\boldsymbol{T}_i$ 为节点类型 Type,300 维向量;$\boldsymbol{V}_i$ 为节点值 Value,1 200 维向量;$\boldsymbol{\omega}_i$ 为节点向量,为 1 500 维向量;m 为 AST 序列长度,即 AST 节点个数.

Token 序列是源代码通过分句、分词得到的一种特殊的文本数据,充分包含源代码的语义信息.定义 Token 序列为 $\varphi=\{\boldsymbol{\varphi}_1,\boldsymbol{\varphi}_2,\cdots,\boldsymbol{\varphi}_n\}$,其中,$n$ 为 Token 序列长度;$\boldsymbol{\varphi}_i$ 为 Token 向量,1 500 维.

与自然语言不同的是,代码语言中存在大量用户自定义的标识符,在模型构建词表时,自定义标识符会使构建的词表过于庞大,占用大量的内存,并且严重影响计算的效率.因此,选择训练集中使用频率最高的 K 个单词构建全局词表,用特殊字符“UNK”表示 OOV.

2.2.2　编码层

在得到由输入层传入的输入向量后,由编码层负责挖掘输入向量深层次的特征.选择 LSTM 模型作为编码器,同时加入残差连接,对于输入的每一个向量 $\boldsymbol{\sigma}_i$,将其编码为 $\boldsymbol{\psi}_i$ 作为编码器的输出.在 LSTM 内部,设置输入序列的固定长度为 50,若序列长度超过固定长度,则需要将输入序列划分为多个片段.在进行训练和测试时,模型存储 LSTM 对上一个输入序列片段编码后得到的隐藏状态,当模型对下一个输入序列片段进行编码时,存储的隐藏状态可以为 LSTM 提供历史上下文信息,使 LSTM 利用缓存的隐藏状态进行下一个输入序列片段的编码,通过利用历史信息增强长距离依赖信息建模的能力.

在试验过程中,随着网络深度的增加,精度会达到饱和,然后迅速退化.退化问题可以通过引入残差结构解决.残差结构是一种线性数学分析的结构,可以在网络深度显著增加时提高准确性.

2.2.3　注意力层

1)上下文注意

神经语言模型无法处理远程依赖[45],加入注意力机制[46-51]可以利用之前的相关隐藏状态,从而使模型能够捕获长距离依赖,有助于预测下一个 Token.在经过 LSTM 编码器计算得到网络输出后,利用上下文窗口中的先前隐藏状态计算隐藏状态之间的注意权重来表示词与词之间的相关程度,称为上下文注意.

注意力层的计算如式(1)~式(8)所示,设置 $\boldsymbol{N}_t=[h_{t-L},\cdots,h_{t-1}]$ 和 $\boldsymbol{O}_t=[x_{t-L},\cdots,x_{t-1}]\in\mathbf{R}^{k\times L}$,分别存储时间步 t 之前 AST 向量和 Token 向量输出的 L 个隐藏状态.模型通过注意力层来计算 h_t 和 $\boldsymbol{N}_t$ 之间的关系得到注意分数 $\boldsymbol{\alpha}_t$,其中 tanh 为非线性激活函数,Softmax 函数用于计算当前隐藏状态 h_t 和 $\boldsymbol{N}_t$ 中 L 个隐藏状态之间的注意权重,表现为概率的形式.根据注意权重重新计算当前时刻的上下文向量 $\boldsymbol{c}_t$. 其中参数 $\boldsymbol{W}^{\mathrm{n}},\boldsymbol{W}^{\mathrm{o}}\in\mathbf{R}^{k\times k}$ 和 $v\in\mathbf{R}^{k}$ 为可训练参数,k 为隐藏状态大小,即 h_t 和 x_t 的维度.

$$\boldsymbol{N}_t=\boldsymbol{N}_t+\boldsymbol{\theta}_t; \tag{1}$$

$$\boldsymbol{O}_t=\boldsymbol{O}_t+\boldsymbol{\varphi}_t; \tag{2}$$

$$\boldsymbol{A}_t=\boldsymbol{v}^{\mathrm{T}}\tanh\left[\boldsymbol{W}^{\mathrm{n}}\boldsymbol{N}_t+(\boldsymbol{W}^{\mathrm{h}}h_t)\,1_L^{\mathrm{T}}\right]; \tag{3}$$

$$\boldsymbol{B}_t = \boldsymbol{v}^{\mathrm{T}}\tanh[\boldsymbol{W}^{\mathrm{o}}\boldsymbol{O}_t + (\boldsymbol{W}^{\mathrm{h}}x_t)\ 1_L^{\mathrm{T}}]; \tag{4}$$

$$\boldsymbol{\alpha}_t = \mathrm{Softmax}(\boldsymbol{A}_t); \tag{5}$$

$$\boldsymbol{\beta}_t = \mathrm{Softmax}(\boldsymbol{B}_t); \tag{6}$$

$$\boldsymbol{l}_t = [\boldsymbol{\alpha}_t;\boldsymbol{\beta}_t]; \tag{7}$$

$$\boldsymbol{c}_t = [\boldsymbol{N}_t\boldsymbol{\alpha}_t^{\mathrm{T}};\boldsymbol{O}_t\boldsymbol{\beta}_t^{\mathrm{T}}]. \tag{8}$$

式中:$\boldsymbol{\theta}_t$ 为节点向量的残差;$\boldsymbol{\varphi}_t$ 为 Token 向量的残差;$\boldsymbol{A}_t$ 为节点向量的注意力值;$\boldsymbol{B}_t$ 为 Token 向量的注意力值;$\boldsymbol{v}^{\mathrm{T}}$ 为可训练参数;1_L 为 L 维的全 1 向量;$\boldsymbol{\beta}_t$ 为 Token 向量的注意力分数;$\boldsymbol{l}_t$ 为节点向量和 Token 向量的连接;$\boldsymbol{W}^{\mathrm{h}}$ 为可训练参数.

2)父注意

除了上下文注意,本文还采用了父注意.上下文窗口中的不同隐藏状态应该与当前预测具有不同程度的相关性.在 AST 中,父节点应该与子节点具有很大的相关性,但是序列化使 AST 失去了父-子信息.为了利用这种结构信息,在进行 AST 序列化时,记录每个 AST 节点的父节点的位置 q,即在该节点之前的节点个数.在时间步 t,模型从 $\boldsymbol{N}_t$ 中检索 1 个父向量 $\boldsymbol{p}_t$,它是父节点位置的隐藏状态,即 h_{t-q}. 父节点信息可以帮助模型做出更准确的预测.

在时间步 t 预测下一个 Token 时,不仅要考虑当前的隐藏状态 h_t,还要考虑上下文向量 $\boldsymbol{c}_t$ 和父向量 $\boldsymbol{p}_t$. 全局 RNN 组件的输出向量 $\boldsymbol{G}_t$ 对下一个 Token 的信息进行编码,将其映射到词汇空间中,然后使用 Softmax 函数生成最终的概率分布 $\gamma_t \in \mathbf{R}^V$.

$$\boldsymbol{G}_t = \tanh(\boldsymbol{W}^{\mathrm{g}}[h_t;x_t;\boldsymbol{c}_t;\boldsymbol{p}_t]); \tag{9}$$

$$\gamma_t = \mathrm{Softmax}(\boldsymbol{W}^{\mathrm{v}}\boldsymbol{G}_t + \boldsymbol{b}^{\mathrm{v}}). \tag{10}$$

式中:$\boldsymbol{W}^{\mathrm{g}} \in \mathbf{R}^{k\times 5k}$ 和 $\boldsymbol{W}^{\mathrm{v}} \in \mathbf{R}^{V\times k}$ 为 2 个可训练矩阵;$\boldsymbol{b}^{\mathrm{v}} \in \mathbf{R}^V$ 为一个可训练的偏差向量;V 为词汇量大小;“;”为连接操作.

2.2.4 输出层

使用 Pointer Mixture Network 模型进行预测,Pointer Mixture Network 模型由 2 个主要组件(全局 RNN 组件和局部指针组件)和 1 个选择器组成.全局 RNN 组件是一个带有注意力机制的 LSTM,从预定义的全局词表中预测下一个 Token,局部指针组件根据学习到的位置权重指向局部上下文中的 Token.选择器生成 1 个标量 $s_t \in [0,1]$ 表示使用全局 RNN 组件的概率,$1-s_t$ 表示使用局部指针组件的概率.

在时间步 t,全局 RNN 组件从全局词汇表中产生下一个 Token 的概率分布 $\delta_t \in \mathbf{R}^V$. 重用注意力分数 $\boldsymbol{\beta}_t$ 表示局部指针组件从局部上下文产生下一个 Token 的概率分布,记为 τ_t. 选择器是一个以当前隐藏状态 h_t 和上下文向量 $\boldsymbol{c}_t$ 为条件的 Sigmoid 函数,其中 $\boldsymbol{W}^{\mathrm{s}} \in \mathbf{R}^{1\times 4k}$ 和 $\boldsymbol{b}^{\mathrm{s}} \in \mathbf{R}^1$ 为可训练权重,$s_t \in [0,1]$ 是平衡 δ_t 和 τ_t 的标量.模型通过连接 2 个概率分布生成最终的预测 y_t.

$$s_t = \sigma(\boldsymbol{W}^{\mathrm{s}}[h_t;x_t;\boldsymbol{c}_t] + \boldsymbol{b}^{\mathrm{s}}); \tag{11}$$

$$y_t = [s_t\delta_t;(1-s_t)\tau_t]. \tag{12}$$

2.3 补全阶段

利用训练阶段得到的 DIBPN 模型对用户输入的代码片段进行 Token 粒度的补全,将待补全代码按照数据集的处理方式进行相同处理,分别获得待补全代码的 AST 序列以及 Token 序列,将二者分别转换为向量的形式作为 DIBPN 模型的输入,使用上一阶段获得的模型进行下一个 Token 预测.当用户使用 IDE 进行开发时,模型根据用户的输入得到最终的概率分布,并将 Token 从序列的形式转化为代码的形式,将概率最高的 5 个或 10 个补全推荐结果展示给用户,用户可以根据需求选择 Token 作为最终的补全结果.

3 试验分析

试验评估前的准备主要包括试验数据集与试验设置,试验过程与结果分析.

3.1　试验数据集与试验设置

3.1.1　试验数据集

使用 JavaScript 数据集(JS)和 Python 数据集(PY)评估不同的方法,从 Github 上获取(http://plml.ethz.ch)并以源码和 AST 序列的形式存储,数据集多次用于之前的研究中[37,52].训练和测试数据集的统计信息如表 1 所示,其中有 100 000 个程序文件用于训练,50 000 个程序文件用于测试.

表 1　数据集

数据集	JS	PY
训练集	1.07×10^8	6.2×10^7
测试集	5.3×10^7	3.0×10^7
Type 词表	95	330
Value 词表	2.6×10^6	3.4×10^6

3.1.2　评估参数

采用代码补全指标准确率(Accuracy)[4,53-54]对所提出的基于指针网络的双输入代码补全模型进行评价.准确率是代码补全中最常见的一个评价指标,表示预测得到正确的补全结果占所有补全操作次数的比例.

$$\text{Accuracy}=\frac{P_{\text{True}}}{P_{\text{Total}}}. \tag{13}$$

式中:P_{True} 为预测正确的 Token 数量;P_{Total} 为预测的全部 Token 数量.

3.1.3　基准模型

为了分析 DIBPN 模型在代码补全中的效果,将 DIBPN 模型与文献[38,40,55]所提出的模型进行对比分析.其中,文献[55]首次使用神经网络技术来解决广泛使用的动态类型编程语言 JavaScript 的代码补全问题.文献[38]第一次将指针网络引入代码补全研究,为代码补全研究提供了一个新的研究方向.文献[40]是在文献[38]的基础上提出 Pointer Mixture Network 模型,该模型将 LSTM 和指针网络联合使用,通过缓解 OOV 问题,进一步提高代码补全的准确性.因此,本文以文献[38,40,55]所提出的模型作为基准模型进行对比分析.

LSTM[55-56]是一个没有任何注意或指针机制的标准 LSTM 网络.采用 LSTM 从序列化的 AST 序列中提取 AST 节点特征,每个 LSTM 单元的输入是先前的隐藏状态和当前 AST 节点的 Type 向量和 Value 向量的拼接,最后一个输出隐藏状态被反馈给一个由单层前馈神经网络和 Softmax 组成的输出层.

Attentional LSTM[40]对 LSTM 进行了改进,是一种包含上下文注意和父注意机制的 LSTM.它计算上下文窗口内的当前隐藏状态与以前所有隐藏状态之间的注意权值生成上下文向量,输出层的输入是当前隐藏状态、上下文向量和当前节点父节点隐藏状态的拼接.

Pointer Mixture Network[38,40]由全局 RNN 组件和局部指针组件构成,可以通过全局 RNN 组件从全局词表中预测下一个 Token,或者通过局部指针组件从局部上下文中预测下一个 Token.

3.1.4　参数设置

本文的基本模型是 1 个单层 LSTM 网络,展开长度为 50,隐藏单元大小为 1 500,节点类型嵌入 300 维,节点值嵌入 1 200 维,Token 嵌入 1 500 维.将初始学习率设为 0.001,衰减率为 0.6,注意力窗口的大小为 50,batch size 为 128,epoch 为 8,dropout 为 0.5,使用的损失函数为 CrossEntropyLoss[40,57].每个试验进行 5 次并计算平均结果.

3.2　试验过程与结果分析

3.2.1　模型在代码补全任务上的效果

为了验证模型在代码补全任务上的效果,将 DIBPN 模型与 Pointer Mixture Network 模型[40]等基准模型进行比较.Pointer Mixture Network 模型首次将神经网络技术与指针网络相结合来进行代码补全,可以通过 2 个不同组件进行预测补全,遵循了 LI 等[40]在 Pointer Mixture Network 模型中使用的相同数据集和类

似的试验设置.

将提出的模型与 3 个基准模型在 2 个数据集上进行比较.对于这 2 个数据集,通过将节点值 Value 的全局词汇表大小分别设为 1×10^3, 1×10^4 和 5×10^4 来创建 3 个特定的数据集.表 2 为不同模型的试验结果.

表 2 不同模型的试验结果 单位:%

模型	JS						PY					
	1×10^3		1×10^4		5×10^4		1×10^3		1×10^4		5×10^4	
	Type	Value	Type	Value	Type	Value	Type	Value	Type	Value	Type	Value
LSTM	77.24	69.90	82.69	75.80	84.94	78.60	71.62	63.60	73.28	66.30	74.72	67.30
Attentional LSTM	79.76	71.70	85.07	78.10	86.16	80.60	73.22	64.90	75.53	68.40	77.83	69.80
Pointer Mixture Network	81.89	73.20	86.84	78.90	87.63	81.00	75.27	66.40	77.83	68.90	78.46	70.10
DIBPN	84.41	75.68	87.80	81.24	88.74	83.42	77.52	68.61	78.43	71.24	80.37	72.67

1)与 LSTM 比较.在 LSTM 中引入 1 个简单的 LSTM 体系结构对程序上下文进行建模,以 AST 作为输入,只考虑源代码的语法结构信息.与 LSTM 相比,DIBPN 不仅考虑源程序的语法结构信息,还考虑源代码的语义信息.如表 2 所示,在每个特定的数据集上,DIBPN 模型的准确率都比 LSTM 高.

2)与 Attentional LSTM 比较.Attentional LSTM 在 LSTM 的基础上进行改进,在 Attentional LSTM 中,计算上下文窗口内的当前隐藏状态与之前所有隐藏状态之间的注意力权重生成上下文向量.输出层的输入是当前隐藏状态、上下文向量和当前节点的父节点隐藏状态三者的结合.与 Attentional LSTM 相比,引入父注意表示节点之间的父-子关系,可以有效缓解因为 AST 序列化引起的部分结构信息丢失.如表 2 所示,相比 LSTM,Attentional LSTM 在 Type 和 Value 预测方面的准确率都有所提升,但与基于指针网络的双输入代码补全模型相比仍有差距.

3)与 Pointer Mixture Network 比较.Pointer Mixture Network 在 Attentional LSTM 的基础上增加了指针网络,可以从全局词表或局部上下文中预测下一个 Token.DIBPN 在此基础上加入 Token 序列作为输入,并增加残差连接弥补信息丢失.结果显示,DIBPN 模型预测 Type 和 Value 的准确率最高分别为 88.74% 和 83.42%,比 Pointer Mixture Network 分别提升 1.11 个百分点和 2.42 个百分点.

3.2.2 模型不同结构的有效性

为了验证各部分的有效性,进行了消融研究.对 AST 节点输入、Token 输入和残差连接 3 种结构进行不同组合的训练,共考虑 8 种情况.为了后续阐述的简便性,将节点向量记为 InAST,Token 向量记为 InToken,在节点向量和 Token 向量加入的残差连接分别记为 R_A 和 R_T.

8 种情况分别如下:

1)InAST+R_A+InToken+R_T:模型 DIBPN 结构,输入包含节点向量和 Token 向量以及二者对应的残差.

2)InAST+R_A:输入包含节点向量及对应的残差,将试验结果与 DIBPN 进行对比,可以体现 Token 向量及对应残差的组合效果.

3)InToken+R_T:输入包含 Token 向量及对应的残差,将试验结果与 DIBPN 进行对比,可以体现节点向量及对应残差的组合效果.

4)InAST+InToken:输入包含节点向量和 Token 向量,不包含残差,将试验结果与 DIBPN 进行对比,可以体现 2 个残差的组合效果.

5)InAST:输入仅包含节点向量,将试验结果与情况 2 进行对比,可以体现节点向量对应残差的单独效果.将试验结果与情况 4 进行对比,可以体现 Token 向量的单独效果.

6)InToken:输入仅包含 Token 向量,将试验结果与情况 3 进行对比,可以体现 Token 向量对应残差的单独效果.将试验结果与情况 4 进行对比,可以体现节点向量的单独效果.

7)InAST+InToken+R_T:输入包含节点向量、Token 向量及其对应残差,将试验结果与 DIBPN 进行对比,可以体现节点向量对应残差的单独效果.将试验结果与情况 3 进行对比,可以体现节点向量的单独效果.

8)InAST+R_A+InToken:输入包含节点向量及其对应的残差和 Token 向量,将试验结果与 DIBPN 进行对比,可以体现 Token 向量对应残差的单独效果.将试验结果与情况 2 进行对比,可以体现 Token 向量的单

独效果.

根据以上 8 种情况,分别在 JS 和 PY 数据集上对模型进行训练,不同结构模型的试验结果如表 3 所示.由表 3 可知:InAST+InToken 组合的试验结果比单独 InAST 或 InToken 更好,证明了 InAST 输入与 InToken 输入对于提高代码补全的准确率是必不可少的.将源代码转换成节点向量有助于充分提取源代码的语法结构特征,转换成 Token 向量能够充分提取源代码的语义特征,有助于细节类 Token 的补全.InAST+InToken 组合输入的效果更优,也证明了二者组合对提高模型准确率是有效的.

与 InAST+R_A, InToken+R_T, InAST+InToken 的结果相比,DIBPN 的准确率最高,证明了 InToken+R_T, InAST+R_A 和 R_T+R_A 组合对于提高代码补全的准确率是有效的;DIBPN 与 InAST+InToken+R_T, InAST+R_A+InToken 结果相比效果更好,证明了单独的 R_A 或 R_T 对提高模型的准确率是有效的.InAST+R_A 比 InAST 的效果好、InAST+R_A+InToken 比 InAST+InToken 的效果好,证明了单独对节点向量使用残差连接是有效的;InToken+R_T 比 InToken 效果好、InAST+InToken+R_T 比 InAST+InToken 效果好,证明了单独对 Token 节点向量使用残差连接是有效的.同时,对节点向量和 Token 向量使用残差连接时的效果提升最明显,在 JS 数据集上,Value 预测最高提升 0.84 个百分点,Type 预测最高提升 0.8 个百分点;在 PY 数据集上,Value 预测最高提升 0.76 个百分点,Type 预测最高提升 0.71 个百分点.残差连接可以使特征信息向上直接传导至上一层激活节点,减少随着网络加深导致的信息丢失,最大可能地保留源代码特征.以上结果表明,加入残差连接有助于源代码信息的保留,可以提高代码补全的准确率.

表 3　不同结构模型的试验结果　　单位:%

模型	JS						PY					
	1×10^3		1×10^4		5×10^4		1×10^3		1×10^4		5×10^4	
	Type	Value	Type	Value	Type	Value	Type	Value	Type	Value	Type	Value
DIBPN	84.41	75.68	87.80	81.24	88.74	83.42	77.52	68.61	78.43	71.24	80.37	72.67
InAST+R_A	83.42	74.57	88.28	80.24	88.83	82.46	76.55	67.65	79.46	70.27	80.45	71.67
InToken+R_T	82.17	73.42	87.04	79.16	88.06	81.26	75.54	66.73	78.28	69.25	78.76	70.45
InAST+InToken	83.61	74.84	88.55	80.47	89.04	82.73	76.81	67.96	79.66	70.55	80.63	71.91
InAST	81.89	73.20	86.84	78.90	87.63	81.00	75.27	66.40	77.83	68.90	78.46	70.10
InToken	75.87	67.70	80.47	73.67	83.36	76.34	69.55	60.93	72.78	63.68	73.44	64.94
InAST+InToken+R_T	83.87	75.05	88.72	80.66	89.34	82.93	77.03	68.16	79.94	70.73	80.85	72.16
InAST+ R_A+InToken	84.08	75.22	88.94	80.94	89.57	83.17	77.28	68.34	80.17	70.98	81.13	72.35

3.2.3　模型不同参数的有效性

在训练过程中进行超参数的调整可以使所提模型的效果达到最佳.本文对以下超参数进行调整:(1)batch size;(2)注意力层数.

1)batch size 调整

batch size 指的是模型迭代 1 次能处理数据的多少,受试验环境的影响,不同批处理大小对模型的影响不同.当 batch size 较小时,数据在设备中的频繁交换可能会存在性能上的缺失;当 batch size 较大时,可以减少训练时间,但所需内存会增加,模型的泛化性能会下降.试验分别设定 batch size 为 64, 128, 256,不同 batch size 的试验结果如表 4 所示.由表 4 可知:batch size 为 128 时的效果最优.

表 4　不同 batch size 的试验结果　　单位:%

batch size	JS						PY					
	1×10^3		1×10^4		5×10^4		1×10^3		1×10^4		5×10^4	
	Type	Value	Type	Value	Type	Value	Type	Value	Type	Value	Type	Value
64	83.25	74.47	88.15	80.07	88.55	82.37	76.43	67.56	79.46	70.03	80.24	71.59
128	84.41	75.68	87.80	81.24	88.74	83.42	77.52	68.61	78.43	71.24	80.37	72.67
256	83.75	74.97	88.65	80.68	89.25	82.95	76.96	68.16	79.87	70.56	80.74	72.18

2)注意力层数调整

将注意力层数分别设定为 1, 2, 3,不同注意力层数的试验结果如表 5 所示.由表 5 可知:增加注意力层数并不能提高模型的性能.当注意力层数增加时,网络的结构越复杂,提取的信息越多,但同时也会把模

型中存在的一些误差进一步放大,从而影响模型的性能.

表 5 不同注意力层数的试验结果 单位:%

注意力层数	JS						PY					
	1×10^3		1×10^4		5×10^4		1×10^3		1×10^4		5×10^4	
	Type	Value	Type	Value	Type	Value	Type	Value	Type	Value	Type	Value
1	84.41	75.68	87.80	81.24	88.74	83.42	77.52	68.61	78.43	71.24	80.37	72.67
2	83.93	75.34	88.95	80.96	89.45	83.16	77.35	68.34	80.16	70.95	81.05	72.33
3	83.15	74.54	88.16	80.04	88.64	82.28	76.42	67.46	79.35	70.13	80.27	71.43

4 结论

1)在 Pointer Mixture Network 的基础上提出 DIBPN 模型,同时将节点向量和 Token 向量作为模型输入,充分结合语法结构信息和语义信息.

2)所提出的 DIBPN 模型在 Token 补全方面取得了比以往先进模型更好的结果.

3)未来将从以下 4 个方面进行扩展:从代码补全准确率和代码补全效率这 2 个方面对模型进行优化,以达到更好的效果;计划寻找更大的数据集对模型进行训练,以提高模型的准确率,同时寻找其他语言的数据集,使模型能够适用更多的编程语言;扩大适用范围,考虑将基于指针网络的双输入结构应用到代码搜索和代码克隆检测等领域;探索代码的其他表示形式,考虑通过代码的其他表示形式进行表征,如程序依赖图;所提出的代码补全目前只适用于通用的编程场景,为特定程序员开展个性化的代码补全研究是一个很好的研究方向,将在后续的研究中进行重点关注.

参考文献:

[1] PLETCHER D M, HOU D Q. BCC: Enhancing code completion for better API usability[C]//2009 IEEE International Conference on Software Maintenance. IEEE, 2009: 393-394.

[2] HOLMES R, MURPHY G C. Using structural context to recommend source code examples[C]//Proceedings of 27th International Conference on Software Engineering. IEEE, 2005: 117-125.

[3] ASADUZZAMAN M, ROY C K, SCHNEIDER K A, et al. CSCC: simple, efficient, context sensitive code completion[C]// 2014 IEEE International Conference on Software Maintenance and Evolution. IEEE, 2014: 71-80.

[4] HOU D Q, PLETCHER D M. An evaluation of the strategies of sorting, filtering, and grouping API methods for Code Completion[C]//2011 27th IEEE International Conference on Software Maintenance (ICSM). IEEE, 2011: 233-242.

[5] LIU F, LI G, ZHAO Y F, et al. Multi-task learning based pre-trained language model for code completion[C]//Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. ACM, 2020: 473-485.

[6] PELSMAEKER D A A, VAN ANTWERPEN H, POULSEN C B, et al. Language-parametric static semantic code completion[J]. Proceedings of the ACM on Programming Languages, 2022, 6: 1-30.

[7] LU S, DUAN N, HAN H, et al. ReACC: a retrieval-augmented code completion framework[C]//Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics. Dublin, Ireland, 2022: 6227-6240.

[8] KYAW H H S, AUNG S T, THANT H A, et al. A proposal of code completion problem for Java programming learning assistant system[C]// Complex, Intelligent, and Software Intensive Systems. Cham: Springer International Publishing, 2019: 855-864.

[9] YANG Y X, CHEN X. Improve language modelling for code completion through learning general token repetition of source code[C]// Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering. 2019: 667-777.

[10] TERADA K, WATANOBE Y. Code completion for programming education based on deep learning[J]. International Journal of Computational Intelligence Studies, 2021, 10(2/3): 78.

[11] FRANKS C, TU Z P, DEVANBU P, et al. CACHECA: a cache language model based code suggestion tool[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015: 705-708.

[12] YANG Y X, CHEN X. Improve language modelling for code completion by tree language model with tree encoding of context (S)[C]//Proceedings of the 31st International Conference on Software Engineering and Knowledge Engineering.

2019: 675-777.

[13] FANG L Y, HUANG Z Q, ZHOU Y, et al. Adaptive code completion with meta-learning[C]//Proceedings of the 12th Asia-Pacific Symposium on Internetware. ACM, 2020: 116-125.

[14] POPOV A, OREKHOV D, LITVINOV D, et al. Time-efficient code completion model for the R programming language[C]//Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021). Stroudsburg, PA, USAACL, 2021: 34-39.

[15] KYAW H H S, FUNABIKI N, KURIBAYASHI M. An implementation of offline answering function for code completion problem in PLAS[C]//2021 IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech). IEEE, 2021: 162-165.

[16] ROOS P. Fast and precise statistical code completion[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015: 757-759.

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

[18] ROBBES R, LANZA M. How program history can improve code completion[C]//2008 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2008: 317-326.

[19] 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.

[20] LEE Y Y, HARWELL S, KHURSHID S, et al. Temporal code completion and navigation[C]//2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013: 1181-1184.

[21] NGUYEN A T, NGUYEN H A, NGUYEN T T, et al. GraPacc: a graph-based pattern-oriented, context-sensitive code completion tool[C]//2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 1407-1410.

[22] OMORI T, KUWABARA H, MARUYAMA K. A study on repetitiveness of code completion operations[C]//2012 28th IEEE International Conference on Software Maintenance (ICSM). IEEE, 2012: 584-587.

[23] 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 Symposium on the Foundations of Software Engineering. ACM, 2009: 213-222.

[24] 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. ACM, 2010: 26-30.

[25] LE K T, RASHIDI G, ANDRZEJAK A. A methodology for refined evaluation of neural code completion approaches[J]. Data Mining and Knowledge Discovery, 2022, 37(1): 167-204.

[26] SCHUMACHER M E H, LE K T, ANDRZEJAK A. Improving code recommendations by combining neural and classical machine learning approaches[C]//Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. ACM, 2020: 476-482.

[27] WANG Y L, LI H. Code completion by modeling flattened abstract syntax trees as graphs[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2021, 35(16): 14015-14023.

[28] YANG K, YU H Q, FAN G S, et al. A graph sequence neural architecture for code completion with semantic structure features[J]. Journal of Software: Evolution and Process, 2022, 34(1): e2414.

[29] KIM S, ZHAO J M, TIAN Y C, et al. Code prediction by feeding trees to transformers[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021: 150-162.

[30] CHIRKOVA N, TROSHIN S. Empirical study of transformers for source code[C]//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, 2021: 703-715.

[31] WANG C Z, HU J H, GAO C Y, et al. Practitioners´ expectations on code completion[EB/OL]. 2023: 2301.03846. https://arxiv.org/abs/2301.03846v1.

[32] XUE B, YI W J, JING F, et al. Complex ISAR target recognition using deep adaptive learning[J]. Engineering Applications of Artificial Intelligence, 2021, 97: 104025.

[33] XUE B, HE Y, JING F, et al. Dynamic coarse-to-fine ISAR image blind denoising using active joint prior learning[J]. International Journal of Intelligent Systems, 2021, 36(8): 4143-4166.

[34] XUE B, HE Y, JING F, et al. Robot target recognition using deep federated learning[J]. International Journal of Intelligent Systems, 2021, 36(12): 7754-7769.

[35] HINDLE A, BARR E T, SU Z D, et al. On the naturalness of software[C]//2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 837-847.

[36] TU Z P, SU Z D, DEVANBU P. On the localness of software[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 269-280.

[37] VINYALS O, FORTUNATO M, JAITLY N. Pointer networks[C]//Proceedings of the 29th International Conference on Neural Information Processing Systems. 2015,2:2692-2700.

[38] BHOOPCHAND A, ROCKTÄSCHEL T, BARR E, et al. Learning Python code suggestion with a sparse pointer network[EB/OL]. 2016: 1611.08307. https://arxiv.org/abs/1611.08307v1.

[39] 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. ACM, 2017: 763-773.

[40] LI J, WANG Y, LYU M R, et al. Code completion with neural attention and pointer networks[C]//Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization, 2018: 4159-25.

[41] ZHANG S, LIN Y F, SHENG H. Residual networks for light field image super-resolution[C]//2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2019: 11046-11055.

[42] CHEN C, LI K L, TEO S G, et al. Gated residual recurrent graph neural networks for traffic prediction[C]//Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence. 2019:485-492.

[43] HEMALAKSHMI G R, SANTHI D, MANI V R S, et al. Deep residual network based on image priors for single image super resolution in FFA images[J]. Computer Modeling in Engineering & Sciences, 2020, 125(1): 125-143.

[44] AI T, LIU Z L, WANG H, et al. Robust vibration signal denoising and diagnosis using encoder-decoder networks with cross-layer residual connection[C]//2021 Global Reliability and Prognostics and Health Management (PHM-Nanjing). IEEE, 2021: 1-5.

[45] CHENG J P, DONG L, LAPATA M. Long short-term memory-networks for machine reading[EB/OL]. 2016: 1601.06733. https://arxiv.org/abs/1601.06733v7.

[46] BAHDANAU D, CHO K, BENGIO Y. Neural machine translation by jointly learning to align and translate[EB/OL]. 2016: 1409.0473. https://arxiv.org/abd/1409.0473.

[47] DEVLIN J, UESATO J, BHUPATIRAJU S, et al. RobustFill: Neural program learning under noisy I/O[C]//Proceedings of the 34th International Conference on Machine Learning. 2017: 990-998.

[48] CHEN X Y, LIU C, SHIN R, et al. Latent attention for if-then program synthesis[C]//Proceedings of the 30th International Conference on Neural Information Processing Systems. 2016: 4581-4589.

[49] NIU Z Y, ZHONG G Q, YU H. A review on the attention mechanism of deep learning[J]. Neurocomputing, 2021, 452: 48-62.

[50] GUO M H, XU T X, LIU J J, et al. Attention mechanisms in computer vision: a survey[J]. Computational Visual Media, 2022, 8(3): 331-368.

[51] KELLER A S, LEIKAUF J E, HOLT-GOSSELIN B, et al. Paying attention to attention in depression[J]. Translational Psychiatry, 2019, 9(1): 279.

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

[53] 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. 2016: 2091-2100.

[54] ALLAMANIS M, BARR E T, BIRD C, et al. Suggesting accurate method and class names[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015: 38-49.

[55] LIU C, WANG X, SHIN R, et al. Neural code completion[C]//Proceedings of the 5th International Conference on Learning Representations (ICLR 2017). Toulon, France, April 24-26, 2017.

[56] SVYATKOVSKIY A, ZHAO Y, FU S Y, et al. Pythia: AI-assisted code completion system[C]//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 2019: 2727-2735.

[57] KINGMA D P, BA J, HAMMAD M M. Adam: a method for stochastic optimization[EB/OL]. 2014: 1412.6980. https://arxiv.org/abs/1412.6980v9.