软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(1):1−26 [doi: 10.13328/j.cnki.jos.007143] [CSTR: 32375.14.jos.007143]

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

预训练模型在软件工程领域应用研究进展*

宫丽娜[1,2], 周易人[1,2], 乔 羽[1,2], 姜淑娟[3], 魏明强[1], 黄志球[1,2]

[1](南京航空航天大学 计算机科学与技术学院, 江苏 南京 211106)

[2](高安全系统的软件开发与验证技术工信部重点实验室 (南京航空航天大学), 江苏 南京 211106)

[3](中国矿业大学 计算机科学与技术学院, 江苏 徐州 221116)

通信作者: 周易人, E-mail: zhouyiren@nuaa.edu.cn; 姜淑娟, E-mail: shjjiang@cumt.edu.cn

摘 要: 近年来深度学习在软件工程领域任务中取得了优异的性能. 众所周知, 实际任务中优异性能依赖于大规模训练集, 而收集和标记大规模训练集需要耗费大量资源和成本, 这限制了深度学习技术在实际任务中的广泛应用. 随着深度学习领域预训练模型 (pre-trained model, PTM) 的发布, 将预训练模型引入到软件工程 (software engineering, SE) 任务中得到了国内外软件工程领域研究人员的广泛关注, 并得到了质的飞跃, 使得智能化软件工程进入了一个新时代. 然而, 目前没有研究提炼预训练模型在软件工程领域的成功和机遇. 为阐明这一交叉领域的工作 (pre-trained models for software engineering, PTM4SE), 系统梳理当前基于预训练模型的智能软件工程相关工作, 首先给出基于预训练模型的智能软件工程方法框架, 其次分析讨论软件工程领域常用的预训练模型技术, 详细介绍使用预训练模型的软件工程领域下游任务, 并比较和分析预训练模型技术这些任务上的性能. 然后详细介绍常用的训练和微调 PTM 的软件工程领域数据集. 最后, 讨论软件工程领域使用 PTM 面临的挑战和机遇. 同时将整理的软件工程领域 PTM 和常用数据集发布在 https://github.com/OpenSELab/PTM4SE.

关键词: 软件仓库挖掘; 预训练模型; 程序语言模型

中图法分类号: TP311

中文引用格式: 宫丽娜, 周易人, 乔羽, 姜淑娟, 魏明强, 黄志球. 预训练模型在软件工程领域应用研究进展. 软件学报, 2025, 36(1): 1–26. http://www.jos.org.cn/1000-9825/7143.htm

英文引用格式: Gong LN, Zhou YR, Qiao Y, Jiang SJ, Wei MQ, Huang ZQ. Research Progress of Pre-trained Model in Software Engineering. Ruan Jian Xue Bao/Journal of Software, 2025, 36(1): 1–26 (in Chinese). http://www.jos.org.cn/1000-9825/7143.htm

Research Progress of Pre-trained Model in Software Engineering

GONG Li-Na[1,2], ZHOU Yi-Ren[1,2], QIAO Yu[1,2], JIANG Shu-Juan[3], WEI Ming-Qiang[1], HUANG Zhi-Qiu[1,2]

[1](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

[2](Key Laboratory for Safety-critical Software Development and Verification (Nanjing University of Aeronautics and Astronautics), Nanjing 211106, China)

[3](School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China)

Abstract: In recent years, deep learning has achieved excellent performance in software engineering (SE) tasks. Excellent performance in practical tasks depends on large-scale training sets, and collecting and labeling large-scale training sets require a lot of resources and costs, which limits the wide application of deep learning techniques in practical tasks. With the release of pre-trained model (PTM) in the field of deep learning, researchers in SE have begun to pay attention to PTM and introduced PTM into SE tasks. PTM has made a qualitative

* 基金项目: 国家自然科学基金 (62202223); 江苏省自然科学基金 (BK20220881); 高安全系统的软件开发与验证技术工信部重点实验室 (南京航空航天大学) 开放项目 (NJ2022027)
收稿时间: 2023-02-06; 修改时间: 2023-06-21, 2023-10-29; 采用时间: 2024-01-04; jos 在线出版时间: 2024-06-18
CNKI 网络首发时间: 2024-06-20

leap in SE tasks, which makes intelligent software engineering enter a new era. However, none of the studies have refined the success, failure, and opportunities of pre-trained models in SE. To clarify the work in this cross-field (pre-trained models for software engineering, PTM4SE), this study systematically reviews the current studies related to PTM4SE. Specifically, the study first describes the framework of the intelligent software engineering methods based on pre-trained models and then analyzes the commonly used pre-trained models in SE. Meanwhile, it introduces the downstream tasks in SE with pre-trained models in detail and compares and analyzes the performance of pre-trained model techniques on these tasks. The study then presents the datasets used in SE for training and fine-tuning the PTMs. Finally, it discusses the challenges and opportunities for PTM4SE. The collated PTMs and datasets in SE are published athttps://github.com/OpenSELab/PTM4SE.

Key words: software repository mining; pre-trained model (PTM); programming language model

随着深度学习技术的发展, 软件工程领域的研究者正广泛应用深度学习 (deep learning, DL) 技术来改进自动化软件工程任务. Watson 等人[1] 通过系统调研软件工程中的深度学习使用技术, 发现当前 70% 以上的软件工程任务正在使用深度学习技术 (比如, 源代码生成[2]、bug 修复[3]、克隆检测[4]等). 然而, 深度学习技术的性能依赖于大规模的可靠数据集, 收集和处理大规模有效数据集往往为研究者带来了相当的挑战, 同时 DL 模型在大规模数据上的训练也为研究增加了工作量和时间成本.

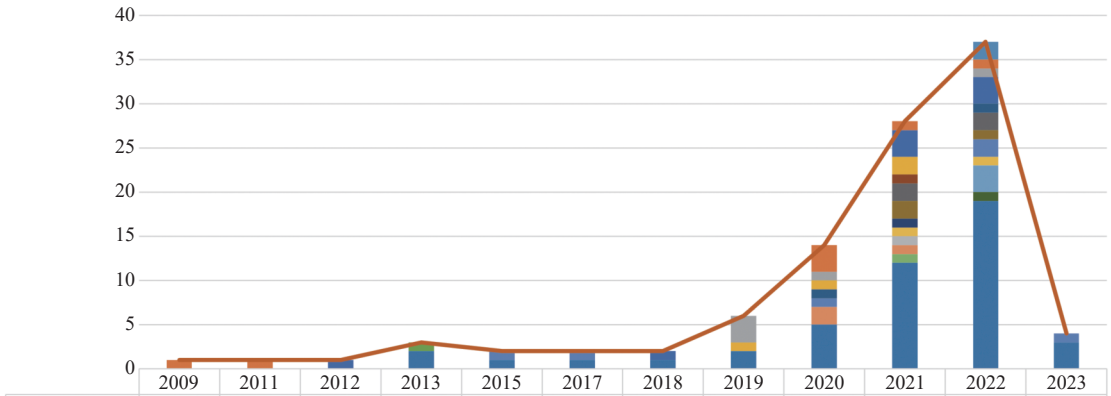
为了避免如此大的花费, 在具体任务中重用预训练模型逐渐成为人工智能领域的共识[5]. 因为预训练模型重用技术采用 PTM (pre-trained model) 作为具体任务的支柱而非重新学习模型, 通过少量数据微调存储丰富知识的预训练模型的方式来解决具体的软件工程任务. 特别是 BERT (bidirectional encoder representations from Transformers)、GPT (generative pre-trained Transformer) 等预训练模型的成功为软件工程领域的任务提供了性能更优越的解决方案, PTM 重用成为软件研究人员和开发者非常关注的研究热点之一.

然而深度学习领域训练 PTM 使用的数据集与 SE 领域数据有很大的差异, 这些差异不能保证深度学习领域 PTM 在不同 SE 任务中都具有很好地可移植性. 更重要地, 目前预训练模型重用技术发展迅速, PTM 在 SE 领域的应用还处于起步摸索阶段, 虽然 Niu 等人[6]对 PTM 在软件工程领域任务进行了总结, 但仅简单调研分析了基于源代码的预训练模型及任务. 目前没有研究提炼各种不同预训练模型在软件工程领域的成功、失败和机遇. 在此背景下, 本文系统梳理了当前软件工程领域基于预训练模型的智能方法相关文献, 从研究框架、常用的预训练模型技术、软件工程领域下游任务及性能评价指标和软件工程领域广泛使用的数据集等多个角度汇总和分析了预训练模型在软件工程领域应用研究.

本文使用 pre-trained models、pre-trained、software engineering 等搜索关键词, 在 Google Scholar、ACM Digital Library、Springer Link online Library、IEEE Xplore Digital Library、Elsevier Science Direct 及 CNKI 在线数据库中全面搜索了截至 2023 年 1 月份发表在期刊和会议上的相关文献. 然后经过人工审查摘要方式去除与 PTM4SE 无关的文献. 接着我们递归地对每篇文献进行正向和反向滚雪球搜索[7], 最终选择出 101 篇高质量的与 PTM4SE 相关文献, 其中期刊论文 17 篇, 会议论文 60 篇, 其他 24 篇.

我们统计分析了其发表出处的分布情况, 如图 1 所示. 可以发现 PTM 在软件工程领域任务中的应用从 2009 年开始, 每年呈增长趋势. 其中不乏在软件工程领域和人工智能领域的国际顶级期刊和顶级会议上的研究论文. 例如: TOSEM 期刊 (4 篇)、TSE 期刊 (3 篇)、ICSE 会议 (8 篇)、ICML 会议 (4 篇)、ASE 会议 (7 篇)、NeurIPS 会议 (1 篇)、SANER 会议 (6 篇)、ACL 会议 (2 篇) 和 OOPSLA 会议 (1 篇). 基于上述分析不难发现, PTM4SE 技术已成为软件工程领域和人工智能领域中一个重要的研究课题. 为了阐明这一交叉领域的工作 (pre-trained models for software engineering, PTM4SE), 需要系统梳理当前基于预训练模型的智能软件工程相关工作.

本文第 1 节对基于预训练模型的智能软件工程方法研究框架进行描述. 第 2 节系统分析软件工程领域常用的预训练模型. 第 3 节给出目前使用预训练模型的软件工程下游任务, 并分析预训练模型在下游任务中的性能. 第 4 节对常用的训练和微调 PTM 的软件工程领域数据集进行总结和分析. 最后总结全文, 并对未来值得关注的研究方向进行探讨.



	2009	2011	2012	2013	2015	2017	2018	2019	2020	2021	2022	2023
ACL	0	0	0	0	0	0	0	0	0	0	2	0
ASE	1	1	0	0	0	0	0	0	3	1	1	0
ESEC/FSE	0	0	0	0	0	0	0	3	1	0	1	0
ICSE	0	0	1	0	0	0	1	0	0	3	3	0
ICML	0	0	0	0	0	0	0	1	1	2	0	0
ISSTA	0	0	0	0	0	0	0	0	1	0	1	0
OOPSLA	0	0	0	1	0	0	0	0	0	0	0	0
NeurIPS	0	0	0	0	0	0	0	0	0	1	0	0
TOSEM	0	0	0	0	0	0	0	0	0	2	2	0
TSE	0	0	0	0	0	0	0	0	0	2	1	0
SIGIR	0	0	0	0	0	0	0	0	0	1	0	0
SANER	0	0	0	0	1	1	0	0	1	0	2	1
IST	0	0	0	0	0	0	0	0	0	1	1	0
ESE	0	0	0	0	0	0	0	0	0	0	3	0
EMNLP	0	0	0	0	0	0	0	0	0	0	1	0
ISSRE	0	0	0	0	0	0	0	0	0	1	0	0
ICSME	0	0	0	0	0	0	0	0	2	1	0	0
UAI	0	0	0	0	0	0	0	0	0	1	0	0
Others	0	0	0	2	1	1	1	2	5	12	19	3
Total	1	1	1	3	2	2	2	6	14	28	37	4

图 1　文献发表出处分布

1　研究框架

开发基于深度学习方法的智能软件工程方法需要昂贵的资源和时间成本, 为了避免如此巨大的成本, 软件工程领域研究者提出了使用预训练模型解决软件工程领域任务的方法, 即基于预训练模型的智能软件工程方法. 该方法是使用少量标签软件工程下游任务数据来训练基于已有预训练模型构建的智能模型, 从而形成最终的 PTM4SE 智能方法, 解决软件工程领域下游任务 (如: 代码生成、程序修复、缺陷报告分类等).

具体的构建过程主要包括 SE 下游任务数据收集及处理、基于预训练模型的智能方法构建、模型训练和模型评价这 4 部分. 图 2 给出了基于预训练模型的智能软件方法研究框架.

(1) 数据收集及处理: 根据软件工程任务从软件工程仓库中收集所需的数据集 (如: 源代码仓库中的代码数据, 缺陷跟踪系统的缺陷报告等), 并对数据集进行标记及预处理, 使其符合模型输入的要求.

(2) 基于预训练模型的智能方法构建: 根据软件工程具体任务从预训练模型池中选取合适的预训练模型 (如: BERT、CodeBERT 或 VGG-16), 并基于该预训练模型构建符合自己要求的智能方法.

(3) 模型训练: 根据 SE 下游任务, 选择合适的预训练模式, 使用收集的少量标签数据来训练模型, 形成高性能的 PTM4SE 模型.

(4) 模型评价: 通过性能评价指标评价基于预训练模型的智能方法在软件工程任务中的性能.

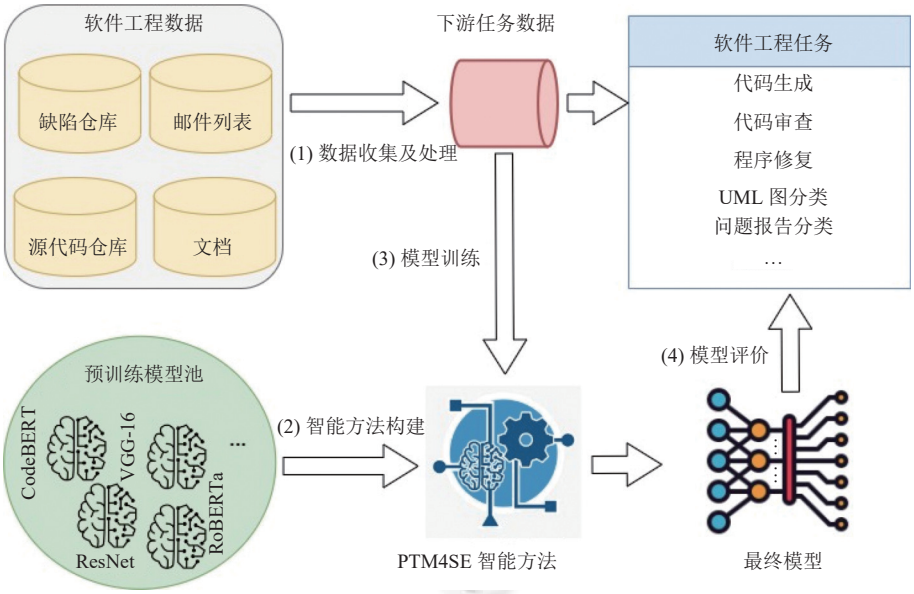


图 2 基于预训练模型的智能软件方法研究框架

我们可以发现根据软件工程下游任务选择合适的预训练模型智能方法及下游任务数据直接决定了基于预训练模型的智能方法的性能. 接下来的章节将从这几方面分别展开详细的叙述.

2 软件工程领域预训练模型及训练范式介绍

预训练模型 (例如, BERT, XLNet, ResNet (residual neural network), VGG (visual geometry group)) 的引入改变了软件工程领域解决任务的方式 (即: 通过使用预训练在大量数据集的模型然后预训练范式来解决软件工程领域下游任务), 提升了软件工程任务的性能. 因此, 预训练模型和预训练范式是基于预训练模型的智能方法的两个重要方面. 近年来, 许多研究者研究了基于不同的预训练模型和预训练范式的智能软件工程方法, 本节将介绍软件工程领域预训练模型和预训练范式两个方面的最新进展.

2.1 预训练模型

2018 年以来, 软件工程领域研究者开始展开基于预训练模型的智能方法研究. 根据训练的预训练模型的数据集来源, 可以将使用的预训练模型分为通用预训练模型 (general pre-trained model)、特定领域预训练模型 (domain-specific pre-trained model) 和源代码预训练模型 (source code pre-trained model).

2.1.1 通用预训练模型

通用预训练模型为人工智能领域中训练在一般领域数据集上的预训练模型, 如自然语言处理 (NLP) 中的训练在英文维基百科或普通新闻数据集的 BERT[8]、GPT[9]和 XLNet[10]模型等, 以及计算机视觉 (CV) 中预训练在 ImageNet 数据集上的 ResNet[11]和 VGG[12]模型等. 目前软件工程领域研究者用于解决软件工程任务的通用预训练模型可以分为自然语言处理和计算机视觉两大类.

(1) 自然语言处理预训练模型

软件工程领域的研究者主要通过使用自然语言处理中的预训练模型来对软件工程领域的文本数据进行很好的编码, 使得下游任务能更好地理解软件工程领域文本中的每个单词. 目前软件工程领域经常使用的自然语言处理中的预训练模型包括: Spacy[13–15]、Stanford CoreNLP[16,17]、Stanford Parser[18]、WordNet[19–23]、BERT[24–43]、XLNet[28,36,38,39]、RoBERTa (robustly optimized BERT approach)[28,30,36,38,39,43–49]、ALBERT (a lite BERT)[28,36,38,39]、DistilRoBERTa (a distilled version of RoBERTa)[30]、DistilBERT[24]、Word2Vec[34,50–52]、GloVe[26,27]、FastText[27]、

NL checker[53]、NLTK[54]和 GPT[42,44,55,56]等.

Zhong 等人[57]和 Rahul 等人[19]在解决从软件库文档的自然语言文本中推断正式规范的任务中, 使用预训练的 POS tagging 识别软件库文档的词性. Zhao 等人[16]在 Stack Overflow 的知识图构建任务中使用 Stanford CoreNLP 工具抽取软件文档的词性及语法树构建, 解决了软件库文档的语料处理问题. Liu 等人[13]在 API 比较任务中使用 Spacy 工具抽取文档的词性及语法树构建. Zhong 等人[53]在识别 API 文档错误的任务中使用 NL checker 工具识别 API 文档中的自然语言错误,包括拼写错误、语法错误、和语义错误等. Pandita 等人[21]利用预训练的 Stanford Parser 工具构建 API 库文档的语法树, 解决移动应用中风险自动识别问题. Howard 等人[20]在处理对通用自然语言文档与软工领域专业词语匹配的任务中提出了自动挖掘软件领域中语义相似词对的方法,实现对 WordNet 功能的增强. Wu 等人[18]在挖掘资源释放规范任务中, 使用 WordNet 进行同义词的识别, 解决代码注释中的语料预处理问题. 此外, Wang 等人[22]和 Liu 等人[23]分别在领域词汇自动构建和程序可读性分析的任务中使用 WordNet 排除普通单词和技术术语以及实现同义词匹配. Ghadhab 等人[24] 将 Google 发布的训练在图书语料库和英文维基百科的 BERT 模型应用于 commit 信息的编码中, 实验发现 BERT 模型能够对 commit 信息中的单词进行很好的特征表示, 训练在使用 BERT 模型编码单词的 DNN 分类器提升了 commits 的分类性能. Biswas 等人[25]同样将预训练的 BERT 模型应用于 Stack Overflow 帖子中句子的情感分类, 实验结果发现 BERT 分类器在软件工程文本的情感分析中取得了可靠的性能. Ghanem 等人[58]使用自然语言处理中预训练的 Word2Vec 模型对公司员工的目标方向文本进行单词嵌入表示, 用以发现有相似工作目标的员工, 帮助人力资源部分减少匹配花费的时间和精力. Perez 等人[55]使用自然语言处理中预训练的 GPT-2 模型提出了一个端到端的 Python 语言生成代码模型, 实验结果表明 GPT-2 模型很好地提高了代码生成任务的性能. Gao 等人[26]使用自然语言处理中的 GloVe 预训练模型的参数来初始化学习 APP 评语中的文本信息表示 CoRe 网络模型的参数, 发现构建的 CoRe 网络能够生成较好的 APP 评语响应生成文本. Pradel 等人[51]使用预训练的 Word2Vec 模型将程序崩溃跟踪中的单词表示为嵌入向量, 实验发现基于生成的单词嵌入表示训练的模型可以有效地在大规模的代码文件中实现 bug 定位. Xie 等人[15]使用 Spacy 处理词性标注和依赖性解析,以识别功能句子中的功能动词和短语模式,并有效提升了匹配 API 查询和功能描述、提高 API 检索的准确性. Liu 等人[54]在自动生成代码 API 摘要的任务中, 采用 NLTK 对描述性文本进行分词、去除停用词、词根还原等预处理工作, 并将 NLTK 用于解析描述性句子, 以识别对 API 概念的引用.

同时, 部分研究者分析了自然语言处理中不同的预训练模型对软件工程领域文本数据的表达能力. 如, Wang 等人[27] 调研了自然语言处理中不同预训练模型 (Word2Vec、GloVe 和 BERT) 对 GitHub 的问题报告信息中单词特征表示对问题报告分类的影响, 发现预训练的 BERT 能更好地适应软件工程领域 GitHub 上的问题报告. Hadi 等人[28]调研了自然语言处理中不同预训练模型 (BERT、XLNet、RoBERTa 和 ALBERT) 对 APP 评论信息中单词特征表示对 APP 评论分类的影响. Al Omran 等人[14]和 Tian 等人[17]分别分析了不同预训练模型的 POS tagging (Spacy, Stanford CoreNLP 等) 对软件库文档数据的表达能力.

(2) 图像处理预训练模型

软件工程领域研究人员使用图像处理的预训练模型来编码表示软件工程领域基于图片的软件工件 (例如, 软件统一的建模语言图 (UML), 图像用户界面 (GUI)), 使软件工程下游任务能很好地理解这些工件. 目前软件工程领域经常使用的图像处理预训练模型包含: VGG[59]、ResNet[60]和 Vision Transformer[61,62]模型. Best 等人[59]使用训练在公共数据集 ImageNet 图片上的 VGG-16 模型编码表示 UML 图, 实验结果发现基于不包括软件工程工件图片的预训练模型, 也可以为 UML 图进行很好的分类, 这为解决软件工程图片工件提供了使用无需定制的现有预训练模型的路径. 接着, Keller 等人[60]将代码通过文本、抽象语法树 (abstract syntax tree, AST) 等形式转化成图片, 然后使用训练在 ImageNet 数据集上[63]的 ResNet 模型来学习代码图片的特征表示, 并在代码克隆检测和漏洞检测任务中取得较好的性能. Soselia 等人[61]将预训练 Vision Transformer 模型应用到基于 webpage 截屏图片的 HTML/CSS 代码生成任务中, 提高了从 GUI 图片逆向生成代码的质量. Li 等人[62]基于预训练 Vision Transformer 模型提出了 Spotlight 方法来提升移动用户界面的理解能力.

综上所述, 现有的预训练模型已经广泛应用于软件工程领域任务中, 特别是基于文本的软件工程任务中. 但是

不同现有预训练模型在不同的 SE 任务中的表现是不同的.

2.1.2 特定领域预训练模型

通用预训练模型在软件工程任务中虽然能够提供较好的预测性能, von der Mosel 等人[29]发现一般领域数据集和软件工程特定领域数据集有很大的不同, 比如在软件工程领域有很多的技术专业术语和一般领域数据集是不同的, 这导致一般领域数据集训练出的预训练模型不能很好地适应软件工程领域的文本特性. 因此, 软件工程领域研究者收集大量的软件工程领域数据集来从零训练深度学习模型形成软件工程特定领域预训练模型, 进而解决软件工程领域任务. 目前特定领域预训练模型主要有 seBERT[29]、text-to-text transfer Transformer (T5) model[64–70]、Word2Vec-SO[50]、BERT-reviews[30]、BERT-SO-1M[30]、BERT-SO-1M-Large[30]、RoBERTa-SO[30]、Sentence DistilBERT[58]、BERToverflow[29,38]、CosSensBERT[38]、DeepMoji[71]和 IR-BERT[72].

von der Mosel 等人[29]使用软件工程领域的 Stack Overflow 帖子, GitHub 的问题报告, 问题跟踪系统 Jira 的问题报告以及 GitHub 的 Git 信息重新训练 BERT 模型得到 seBERT 特定领域模型, 他们比较了预训练在软件工程领域数据集的 seBERT 和自然语言处理的 BERT 模型, 发现使用软件工程数据进行预训练是有价值的. Tufano 等人[64]使用 Stack Overflow 和 CodeSearchNet 的 Java 源代码及软件领域文本数据集重新训练了特定领域的 T5 模型, 并将模型通过微调的方式应用到自动化代码审查中, 提高了代码审查的性能. 同样, Prenner 等人[30]使用软件工程领域数据集重新训练 BERT 及 RoBERTa 模型, 提出了 BERT-reviews, BERT-SO-1M, BERT-SO-1M-Large, RoBERTa-SO 特定领域模型, 并将模型应用于情感分析、APP 评论分析、SO 评论分类等任务, 发现预训练的模型能够提供更好的性能.

2.1.3 源代码预训练模型

软件工程领域的源代码数据 (如: Python/Java 程序等) 与自然语言处理中的文本数据或软件工程领域文本数据有很大的区别, 为了能更好地捕获源代码数据中的语法和语义信息, 软件工程领域的研究者收集大量的源代码数据集来重新训练深度学习模型形成软件工程领域的源代码预训练模型. 目前基于源代码数据的预训练模型主要有 CC2Vec[31]、code2vec[31,73]、CodeT5[32]、CodeBERT[29,30,33,35,44–46,74–83]、GraphCodeBERT[44–46,75,77,80–86]、CodeBERTa[30,35]、C-BERT[33]、CuBERT[81,85,87,88]、PLBART[44,75,81,89]、OSCAR[76]、InferCode[90]、DOBF[77]、SynCoBERT[44,83,91]、CLawSAT[92]、CODE-MVP[81,86]、CodeTrek[93]、COMBO[65]、Codex[42,56,69,82,94–97]、SCodeR[98]、Corder[99]、SPT-Code[100]、TreeBERT[85]、UniXcoder[44,82,83,98]、CugLM[80]、StructCoder[101].

上述模型中, CC2Vec, code2vec, CodeT5, CodeBERT, GraphCodeBERT, CodeBERTa, C-BERT, CuBERT, PLBART, Codex, SPT-Code, TreeBERT 和 UniXcoder 模型是软件工程领域研究者将深度学习通用预训练模型 (例如, Word2Vec, T5, BERT, GPT) 重新训练在他们从 GitHub 等开源项目收集的源代码数据得到的源代码预训练模型. 如 CodeBERT[74]源代码预训练模型是 Feng 等人将从 GitHub 收集的包括 Python、Java、JavaScript、PHP、Ruby 和 Go 这 6 种语言的 8.3 MB 的源代码数据集训练 BERT 模型得到的 CodeBERT 源代码预训练模型. Guo 等人[84]认为现有的源代码预训练模型将代码片段视为标记序列, 会忽略代码的固有结构 (即代码语义), 提出了一种考虑代码固有结构的编程语言预训练模型 GraphCodeBERT, GraphCodeBERT 模型在训练阶段使用数据流结构, 实现了代码中包含的固有结构学习. Niu 等人[100]提出了源代码联合预训练模型 SPT-Code, 通过引入了 3 种联合预训练任务来学习源代码序列、代码结构 AST 序列及代码的自然语言描述信息. Jiang 等人[85]提出了一种基于树的预训练模型 TreeBERT, 其把代码对应的 AST 表示为一组组合路径, 通过树掩码语言建模 (TMLM) 和节点顺序预测 (NOP) 任务来进行训练. Guo 等人[44]提出了 UniXcoder 预训练模型, 其通过一对一映射方法将源代码的 AST 转换为并行序列结构, 以保留树中的所有结构信息, 实现更全面的代码表示.

OSCAR[76], InferCode[90], SynCoBERT[91], CLawSAT[92], CODE-MVP[86], CodeTrek[93], COMBO[65], SCodeR[98], Corder[99]和 DOBF[77]是软件工程领域研究者通过源代码数据集训练自己构建的深度学习模型得到的预训练模型. 如 OSCAR[76]预训练模型是 Peng 等人提出的一种基于 Transformer 的层次化预训练模型, 通过捕获代码长序列之间的上下文信息实现源代码语义的理解. 其是基于一种新的代码表示学习范式, 不仅从源代码的中间表示 (IR) 中学习基本操作的语义表示, 更从静态程序分析得到的抽象解释 (即通过数学上的特征描述程序的可能行为) 中学

习环境转换语义表示. InferCode[90]预训练模型是 Bui 等人提出的基于树卷积神经网络 (TBCNN) 的生成模型, 其对代码片段通过随机地用特殊标记符号替换, 及将它们移动到序列的末尾实现具有双向上下文的代码填充, 进而解决实际代码编写过程中通过多次编辑和优化产生代码, 而非从左到右的顺序编写代码问题. DOBF[77] 预训练模型是 Lachaux 等人提出的以还原混淆函数的原始源代码形式为目标的预训练模型. 其以将被替换为无信息名称的函数和变量名的混淆函数恢复回原始形式为训练目标的序列到序列模型, 以实现对源代码语义的更好表示. 同时, 软件工程研究者将对比学习引入到软件工程任务中构建了 SynCoBERT[91], CODE-MVP[86], COMBO[65], SCodeR[98]和 Corder[99]等源代码预训练模型, 实验结果表明在软件工程实际任务中这些模型比直接使用通用预训练模型重新训练的模型更能得到更好的性能. 如 Wang 等人[91]为了进一步探索编程语言的特性, 提出了一种语法引导的多模态对比训练方法 SynCoBERT, 以标识符预测和 AST 边预测为目标, 使用多模态对比学习策略学习代码语义中的互补信息, 实验结果表明 SynCoBERT 在下游任务中优于 CodeBERT 和 GraphCodeBERT 方法.

综上所述, 目前软件工程领域研究者更多地聚焦在自然语言处理中的预训练模型, 其他领域的预训练模型 (如图像处理、语言处理) 等还比较少. 通过 Best 等人[59]和 Keller 等人[60]对图像处理中的预训练模型挖掘可以发现, 研究人员可以深入挖掘图像处理领域的预训练模型对软件工程领域任务的影响. 图 3 给出了目前软件工程领域文献中使用的预训练模型的文献分布情况.

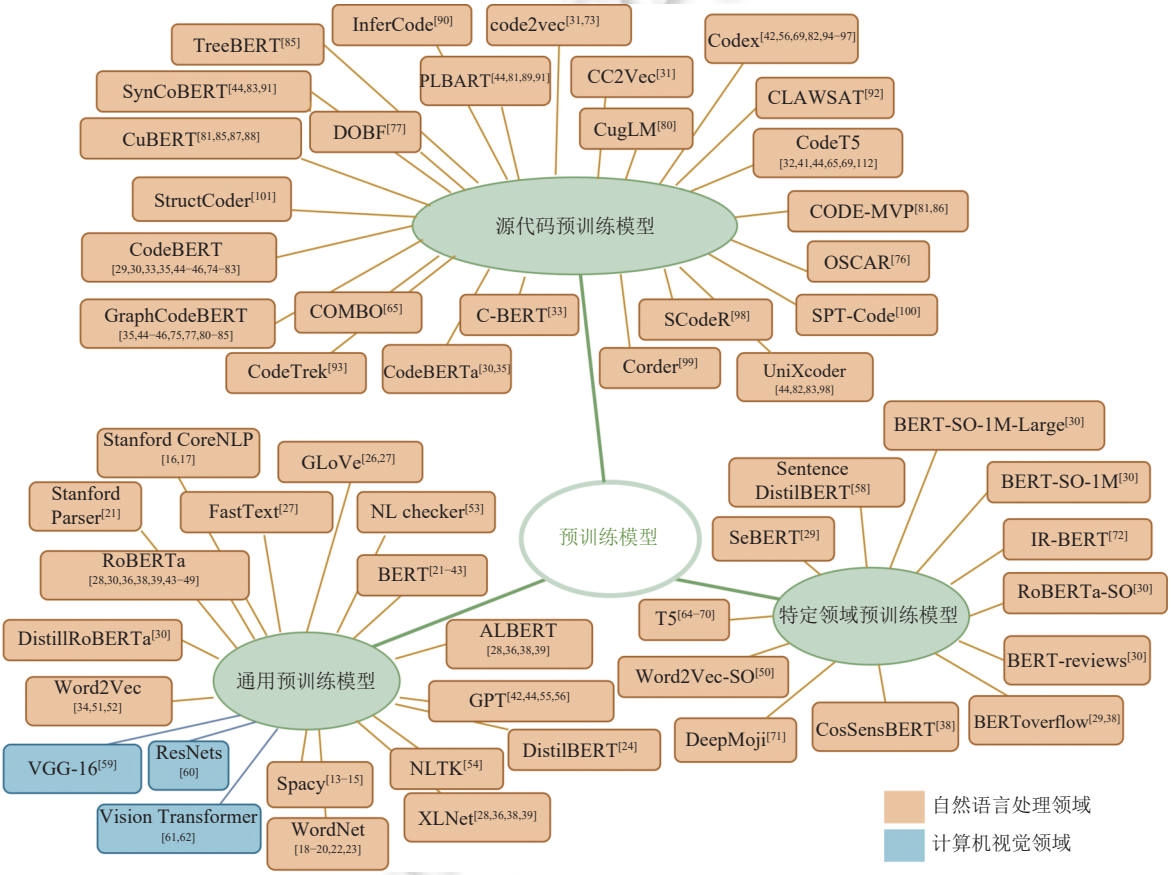


图 3 软件工程领域文献中使用的预训练模型分布情况图

2.2 预训练范式

软件工程领域研究者根据目标任务的不同, 选择不同的预训练范式构建基于预训练模型的智能软件方法, 当前主流的预训练范式主要包括预训练-特征表示和预训练-微调两种范式. 经过统计, 101 篇文献中存在 71 篇文献

采用了预训练-微调范式, 另外 30 篇采用了预训练-特征表示范式.

2.2.1 预训练-特征表示

预训练-特征表示是软件工程领域研究者直接使用训练在大规模数据集上的预训练模型来对软工领域的任务数据集进行编码特征表征, 然后通过编码的特征表征来构建较好的分类器等模型实现软件工程领域任务. Ghadhab 等人[24]直接使用自然语言处理中现有的 BERT 预训练模型来表征细粒度代码更改的特征, 并使用这些特征表示数据集来训练深度学习网络 (DNN) 对代码提交进行分类. 同样, Tian 等人[31]使用自然语言处理的 BERT 预训练模型来表征细粒度代码更改的特征, 并使用这些特征表示数据集来训练逻辑回归分类器对代码提交是否为修复提交进行分类.

2.2.2 预训练-微调

预训练-微调预训练范式是软件工程领域研究者以训练在大规模数据集上的预训练模型为基础构建包含下游任务的智能模型, 然后通过少量下游任务数据集对构建的智能模型进行微调训练, 最终构建适应下游任务的智能模型. 这种模式避免了针对不同任务需要大量数据集从头训练模型. 预训练-微调范式中根据微调的范围又可分为只微调任务层和微调整个智能模型两类.

微调任务层是冻结预训练模型的特征表示层, 通过少量的下游任务数据只微调模型的任务层参数. Best 等人[59]提出了基于训练在 ImageNet 数据集上的 VGG 预训练模型的 UML 分类模型, 通过实验发现冻结 VGG 的特征表示层, 使用软件工程领域的 UML 图片数据微调最后分类层的预训练范式可以得到很好地性能.

微调整个智能模型是使用下游任务数据集微调基于预训练模型的智能模型包括预训练模型的参数来得到最终的智能模型. Hadi 等人[28]提出了在预训练 BERT 模型基础上加入一个包含前馈密集层和 Softmax 激活函数的分类层的 APP 评论分类智能模型, 并通过收集的数据集来微调整个智能模型, 提升了 APP 评论分类的性能. 同样, Zhang 等人[36]为了解决软件工程文本信息的情感分类问题, 在预训练 BERT 模型基础上加入了一个包含 前馈密集层和 Softmax 激活函数的分类层的情感分类智能模型, 通过收集文本数据集微调整个智能模型. De Bortoli Fávero 等人[34]对 BERT 模型的上下文嵌入进行微调, 在软件需求语料库上训练后, 有效提升模型在软件工作量评估任务上的效果. Jung 等人[78]以预训练的 CodeBERT 模型作为编码器提出了 CommitBERT 模型, 通过收集的数据集微调整个 CommitBERT 模型解决代码提交信息生成任务.

目前软件工程领域应用较多的是预训练-微调预训练范式, 通过微调后的模型能更好地适应软件工程领域数据集的特征. 但是当微调数据集过少时也可能会出现模型过拟合使用的微调的数据集, 降低模型泛化能力. 因此, 还需要研究者进一步研究预训练模型和软件工程下游任务之间的对应关系.

3 基于预训练模型的软件工程领域下游任务

预训练模型的强大学习能力使得软件工程领域研究者将预训练模型应用到各种软件工程任务中. 在本节中, 我们将软件工程领域的下游任务根据输入数据类型分为程序语言 (program language, PL) 相关任务, 软件工程领域自然语言 (natural language, NL) 相关任务, 程序语言与自然语言交互任务和软件工程领域图像相关任务. 下面将分别从任务描述及相应任务上的预训练模型性能分析评价来描述每种任务.

3.1 程序语言 PL 相关任务

3.1.1 任务描述

开发人员使用程序语言编写的源代码质量决定了软件系统的质量. 因此, 软件工程领域研究者将源代码中蕴藏的丰富信息应用于不同软件工程领域相关任务中, 如代码片段分类[24,29,30,32–35,38,44–46,60,65,75–77,86–88,90–93,102–108]、程序修复[30–32,35,47,49,66–68,79,87,88,97,100]、代码补全[44,80,92,100,109–111]、语言迁移[32,75,77,89,100,101,112]、API 推荐[15,52]和其他[15,52,113]等, 提升软件系统的质量.

(1) 代码片段分类 (code snippets classification)

代码片段分类任务是通过基于预训练模型的智能方法捕获代码片段中丰富的语法和语义信息来预测代码片

段的类型, 如: 算法类型的分类[45,65,76]、缺陷分类[31,32,86,91,93,102]、技术债务分类[30,103]和漏洞分类[33,46,104,105]等, 为开发者提供有用的分类信息, 帮助开发者更好理解代码.

针对代码片段的算法类型分类, Peng 等人[76]首先使用自己提出的 OSCAR 预训练模型对学生在 OJ 平台写的 C/C++代码 POJ104 进行算法分类, 与以往的方法相比, 预训练模型 OSCAR 方法大大降低了算法分类的错误率. 接着, Wang 等人[45]使用预训练的 RoBERTa 和 CodeBERT 模型对 POJ104 数据集的算法进行分类, 预训练模型大大提升了算法分类的精确度和 mAP (mean average precision).

针对代码片段的缺陷分类, 研究者根据代码片段的粒度, 可以分为细粒度的代码提交的分类或粗粒度的源代码分类. 如, Tian 等人[31]使用 BERT、code2vec 及 CC2Vec 预训练模型来预测细粒度代码提交的正确性. Wang 等人[32]使用 CodeT5 预训练模型来预测源代码的正确性 (即是否包含缺陷).

针对代码片段的技术债务的分类, 研究者通过预训练模型学习的代码片段的语法和语义信息来判断代码片段中是否包含技术债务. 如, Prenner 等人[30]使用预训练在不同数据集上的 BERT 模型来解决代码片段的技术债务检测问题, 发现预训练模型的有效性. Liu 等人[103]使用预训练模型开发了检测代码片段技术债务的工具.

针对代码片段的漏洞类型的分类, 研究者通过使用预训练模型来学习代码片段的特征表示来判断代码片段是否存在漏洞. 如, Buratti 等人[33]使用预训练模型来判断代码片段中是否存在漏洞, 发现漏洞检测的准确率得到很大的提升. Wu 等人[104]利用预训练模型提出了智能合约的漏洞检测工具 Peculiar, 使得漏洞检测的精确度和召回率都得到了很大的提升. Hanif 等人[46]通过使用开源 C/C++漏洞代码数据集预训练 RoBERTa 模型得到了 VulBERTa 安全漏洞检测模型, 实现源代码中的漏洞检测.

(2) 程序修复 (program repair)

程序修复任务[114]是采取不同的技术自动生成修复存在缺陷的代码片段. 为了提高程序修复的性能, 软件工程领域部分研究者利用预训练模型来学习成对的代码片段的语法与语义信息, 并将该信息应用到程序修复任务中. Mashhadi 等人[79]提出了基于 CodeBERT 预训练模型的自动程序修复方法, 可以生成不同长度的修复程序, 同时可以修复不同类型的缺陷. Mastropaolo 等人[66]和 Berabi 等人[67]将预训练的 T5 模型应用于程序修复任务中, 实验结果表明 T5 预训练模型可以利用自监督训练阶段挖掘更多的额外信息, 进而提升程序修复的性能. Yuan 等人[68]针对目前的程序修复方法中只能为单一编程语言生成补丁问题, 提出了基于 T5 的具有跨多种编程语言 (C/Java/JavaScript/Python) 持续学习能力的 CIRCLE (continual repair across programming languages) 程序修复框架, 实验结果表明了 CIRCLE 方法可以在持续学习设置中有效且高效地修复多种程序语言. Ahmed 等人[47]基于编译器的诊断以及 RoBERTa 强大的学习能力提出了 SynShine 工具, 实现程序的自动化语法修复, 实验结果表明了 SynShine 工具在 Java 编程语言中的自动语法修复能力.

(3) 代码补全 (code completion)

代码补全任务是基于上下文代码信息实时建议下一个可能的符号 (例如, 类名、方法名) 来补全代码片段, 加速软件的开发. 研究者将自然语言处理的预训练模型引入到代码补全任务中捕获上下文代码规律. 如 Liu 等人[80]提出了一个基于多任务学习的预训练语言模型用于解决代码补全的任务, 在补全过程中, 可以采用多任务学习来联合预测符号及其类型, 同时也可以利用预测的类型来辅助符号的预测, 实验结果表明了预训练模型的有效性. Ding 等人[109]针对目前的基于预训练模型的代码补全方法中没有考虑同一项目中其他文件中的丰富语言, 即跨文件上下文问题, 提出了 CoCoMIC 方法, 是基于预训练模型开发的联合文件内和跨文件上下文信息的框架, 实验结果表明 CoCoMIC 可以改进现有的基于预训练模型的代码补全方法的性能. Gong 等人[110]提出了 MultiCoder 的多编程语言尤其是低资源编程语言预训练模型方法, 其通过在 MultiPL 预训练中引入 MultiPL 混合专家 (MoE) 层来增强低资源代码完成. 实验结果表明了提出的方法在代码补全中的有效性. Lu 等人[111]针对只关注文件或项目内代码上下文信息的问题, 提出了一种利用外部上下文的检索增强代码补全框架, 其利用词法复制和检索引用具有相似语义的代码. 实验结果表明了提出的方法在 Python 的 Java 语言代码补全任务中的有效性.

(4) 程序语言迁移 (program translation)

程序语言迁移任务即为程序语言翻译任务, 是利用模型将源语言编写的代码作为输入翻译为等效的目标语言

编写的代码, 翻译后的代码语义应该与输入的代码完全匹配. 因此, 部分研究者将预训练模型引入到程序语言迁移任务中, 提升程序语言迁移任务的性能. 如, Ahmad 等人[75]提出了 PLBERT 预训练模型, 该模型预训练在大量的去噪的 Java 和 Python 程序语言中, 能够很好地学习捕获特征的区别性信息, 提高程序语言迁移任务的性能. 同样, Lachaux 等人[77]提出了 DOBF 的预训练模型, 并应用于 Java 语言与 Python 语言之间的迁移任务, 发现 DOBF 可以很好地提升语言翻译的性能. Tipirneni 等人[101]将结构感知引入编码器-解码器 Transformer 模型 StructCoder, 其通过源代码的语法树和数据流图使编码器具有结构感知能力, 通过引入 AST 路径预测和数据流预测任务确保解码器保留目标代码的语法和数据流. 实验结果表明对目标语法和数据流建模的 StructCoder 可以提升程序语言迁移的性能.

(5) API 推荐 (API recommendation)

API 推荐任务是根据开发人员实际需要来自动化地推荐给开发人员合适的 API 序列, 满足开发人员实际需求. Sun 等人[52]基于预训练的 Word2Vec 模型提出了 ReqLib 方法, 根据项目实际需求, 推荐给开发者需要的 API, 通过实验表明 ReqLib 方法在 API 推荐中的有效性.

3.1.2 性能分析评价

预训练模型在程序语言相关任务中得到了很好的应用, 并取得了较满意的性能. 同时, 为了更全面地展示程序语言相关任务中常用的预训练模型及其性能, 本节统计分析了程序语言相关任务中常用预训练模型及其在相应任务中达到最高性能, 如表 1 所示, 其中, 加粗数据为各项任务中指标的最优值.

表 1 程序相关任务的性能比较和统计

PL任务	子任务	PTM	Accuracy	Precision	Recall	*F*1	mAP	BLEU	EM	CodeBLEU	MCC	EditSIM	Number of fixed bugs
代码片段分类	代码提交分类	BERT	**0.80**	**0.84**	**0.75**	**0.79**	—	—	—	—	—	—	—
	算法类型分类	CodeBERT	—	0.85	—	—	**0.83**	—	—	—	—	—	—
		ResNet50	**0.90**	**0.86**	**0.87**	**0.86**	—	—	—	—	—	—	—
	技术债务检测	BERT	—	—	—	**0.82**	—	—	—	—	—	—	—
	漏洞检测	GraphCodeBERT	—	**0.92**	**0.92**	**0.92**	—	—	—	—	—	—	—
		CuBERT	**0.72**	—	—	—	—	—	—	—	—	—	—
		ResNet50	—	0.91	0.91	0.91	—	—	—	—	—	—	—
	缺陷预测	CodeT5	0.66	—	—	**0.60**	—	—	—	—	**0.27**	—	—
		CodeBERTa	0.70	—	—	0.59	—	—	—	—	0.27	—	—
		CuBERT	**0.95**	—	—	—	—	—	—	—	—	—	—
	克隆检测	SynCoBERT	—	0.97	**0.98**	**0.97**	0.88	—	—	—	—	—	—
		SCodeR	—	0.95	0.96	0.95	**0.92**	—	—	—	—	—	—
		UniXcoder	—	**0.98**	0.93	0.95	0.91	—	—	—	—	—	—
		GraphCodeBERT	**0.97**	0.97	0.97	**0.97**	—	—	—	—	—	—	—
程序修复	—	CodeT5-base	—	—	—	—	—	0.77	**0.22**	—	—	—	—
		GraphCodeBERT	—	—	—	—	—	**0.80**	0.17	—	—	—	—
		CuBERT	**0.89**	—	—	—	—	—	—	—	—	—	—
		T5	0.60	—	—	—	—	—	—	—	—	—	**64/182**
		PLBART	—	—	—	—	—	0.77	0.19	—	—	—	—
		CodeBERTa	—	**0.82**	**0.81**	**0.81**	—	—	—	—	—	—	—
代码补全	—	CugLM	—	—	—	—	—	—	**0.84**	—	—	—	—
		UniXcoder	—	—	—	—	—	—	0.43	—	—	**0.72**	—
		PLBART	—	—	—	—	—	—	0.38	—	—	0.68	—
程序语言迁移	—	StructCoder	—	—	—	—	—	0.85	**0.67**	**0.88**	—	—	—
		SPT-Code	—	—	—	—	—	**0.90**	0.64	—	—	—	—
		PLBART	—	—	—	—	—	0.83	0.65	**0.88**	—	—	—
API推荐	—	Word2Vec	—	**0.23**	**0.93**	—	—	—	—	—	—	—	—

从表 1 中可以得出以下结论.

• 代码片段分类任务经常使用的性能评价指标主要有: Accuracy, Precision, Recall, *F*1, AUC、MCC (Matthews correlation coefficient score) 等. 这些性能指标值越大, 模型在代码片段分类任务中越好.

• 程序修复任务常用的性能评价指标主要有: exact match (EM)、number of fixed bugs 和 BLEU (bilingual evaluation understudy) 等. EM 用来衡量生成的修复代码是否与开发人员实际实现的修复代码完全相同. Number of fixed bugs 是通过运行测试用例来查看生成的补丁是否通过测试, 进而得到修复的 bugs 的个数. BLEU 用来计算预测和正确答案之间的 n-gram 相似度, 为 n-gram 匹配精度分数的几何平均值. EM, BLEU 和 number of fixed bugs 越大, 模型越好.

• 代码补全任务常用的性能评价指标主要有: EditSIM (edit similarity)、EM (exact match)、Perplexity 等. 其中, Perplexity 是 token-level 代码补全的评估指标, 用来度量模型预测样本的好坏程度, 即下一个词时的平均可选择数量. 而 EM 和 EditSIM 是 Line-level 代码补全的评估指标. EditSIM 是两个单词之间 Levenshtein 距离, 即将一个单词更改为另一个单词所需的最小单字符编辑次数, 包括插入、删除或替换. EM 用来评估模型预测中匹配到正确答案的百分比. 通常是 Perplexity 和 EditSIM 越小, 模型越好. Accuracy 和 EM 越大, 模型越好.

• 程序语言迁移任务常用的性能评价指标主要有: BLEU、EM 和 CodeBLEU 等. 其中 CodeBLEU 是除了 n-gram 匹配之外, 还考虑了基于代码结构的句法和语义匹配的评价指标. BLEU, EM 和 CodeBLEU 越大, 模型越好.

• 程序 API 推荐任务常用性能评价指标主要有: Precision 和 Recall 等.

同时, 我们发现, 在程序语言相关任务中, 常用的预训练模型是源代码预训练模型中的 CodeBERT、GraphCodeBERT、CuBERT、CodeT5、UniXcoder、PLBART、CodeBERTa、SynCoBERT、StructCoder 等, 而通用预训练模型中主要是 BERT 和 ResNet50, 这正说明源代码预训练模型较与通用预训练模型来说, 更能很好地表示源代码特征.

但同时我们发现, 相同的预训练模型在不同的 SE 程序相关任务中的表现是不一样的. 如在克隆检测中, GraphCodeBERT 有较好的表现能力, 在代码补全中 UniXcoder 有较好的表现能力. 另外, 我们还发现针对特定任务的预训练模型要比通用预训练模型的微调得到更好的性能. 因此, 软件工程领域研究者还需要进一步研究针对下游任务的预训练模型选择问题, 从而给软件工程领域研究者和实践者一个很好的指引.

3.2 自然语言 NL 相关任务

3.2.1 任务描述

软件开发和维护过程中除了有程序语言编写的代码外, 还会包括描述开发和维护过程的自然语言文本, 如描述软件项目问题的问题报告、提交代码改变的描述、APP 的评论等. 这些自然语言的文本信息对挖掘软件开发经验, 提高软件质量也有很重要的作用. 所以软件工程领域研究者针对这些文本数据展开了研究. 同时, 相较于程序代码, 软件工程领域的自然语言文本与自然语言处理中的文本更相近. 因此, 研究者将自然语言处理中的预训练模型引入到软件工程领域自然语言相关的任务中, 如软件工程相关文本分类任务 (例如, 问题报告分类、APP 评论分类、文本情感分类等)[25,27–30,35–38,48,71]、评论回复自动生成[26,43,69]、跟踪链接发现[40]等任务.

(1) 软件工程相关文本分类 (software engineering-related text classification)

软件工程相关文本分类任务是通过基于预训练模型的智能方法捕获软件工程相关文本 (例如, 问题报告、代码提交及评论等) 的编码表示来预测文本的类型, 如问题报告的分类[27,29,48]、APP 评论分类[28,30,39]和文本情感分类[25,29,36,39,71]等, 为开发者提供有用的分类信息, 帮助开发者更好地理解软件工程领域的文本信息.

上述任务中, 问题报告分类任务是利用不同技术对问题跟踪系统问题报告的类型和优先级进行分类. 问题报告是用户在使用软件系统过程中发现问题的自然语言描述, 随着自然语言处理中预训练模型的提出, 研究者将预训练模型引入问题报告分类任务中, 提升问题报告分类的性能. Wang 等人[27]调研了自然语言处理中预训练模型 BERT 在 GitHub 问题报告中的分类能力, 发现使用软件工程特定领域数据集微调后的 BERT 能够改进问题报告的分类能力. 但若是自然语言处理中的预训练模型, 需要大量训练集, BERT 才能表现出较好的性能. 同样, von der Mosel 等人[29]使用软件工程领域的数据集重新训练了 BERT 模型提出了 seBERT 预训练模型, 并将其应用于问题报告分类任务中, 提高了问题报告分类的性能. Izadi 等人[48]将自然语言处理中的 RoBERTa 预训练模型引入到问

题报告类型和优先级分类任务中, 实验结果发现微调后的 RoBERTa 可以使分类精度达到 82%.

APP 评论分类任务是采用不同技术对应用程序 (例如, Google Play Store, Apple APP Store 和 Twitter data) 中的评论进行分类 (例如, 问题报告, 功能需求等), 这些信息对应用程序开发者来说, 都是宝贵资源, 可以应用到软件工程的许多领域 (例如, 需求工程、测试等). 由于传统方法依赖于手动标记的数据集, 这限制了模型的分类能力, 部分研究者将预训练模型引入到 APP 评论分类中, 提高 APP 评论的分类能力. 如, Hadi 等人[28]挖掘了不同的预训练模型在 APP 评论分类中的应用, 发现与之前的方法相比, 预训练模型在某些情况下是最好的选择, 当需要更高的性能和更低的预测时间时, 使用特定领域预训练模型是值得考虑的.

文本情感分类任务是对软件工程领域相关文本 (例如, Stack Overflow 中的帖子, 代码评审评语等) 的情感进行分类, 是开发者或用户对软件系统工件的观点、态度或情绪, 在软件工程领域得到广泛的关注, 提出了各种不同的方法. 随着深度学习领域预训练模型的提出, 部分研究者将预训练模型引入到软件工程领域相关文本情感分类中, 进一步提升了情感分类的性能. 如, Zhang 等人[36]基于 4 种预训练模型 (BERT、RoBERTa、XLNet 和 ALBERT), 通过预训练-微调的模式对文本情感的分类能力进行了评估, 发现预训练模型比现有的方法在 $F1$ 性能上高出 6.5%–35.6%. Chen 等人[71]基于文本信息中的表情符号信息提出了 SEntiMoji 模型, SEntiMoji 模型是基于预训练在 Tweets 上的 DeepMoji 模型来学习软件工程领域的情绪感知表征, 提升文本情感分类的性能.

(2) 评论回复自动生成 (review response generation)

评论回复自动生成任务是针对用户对应用程序的评论准确自动化地给出相应回复文本的任务. 准确的回复应用程序评论是缓解用户的担忧, 改善用户体验的方法之一. 软件工程领域的研究者已将自然语言处理中的预训练模型技术引入到评论回复生成任务中, 提高了回复自动生成任务的准确度. 如 Gao 等人[26]提出了结合自然语言处理中的 BERT 预训练模型的 CoRe 方法, 该方法自然合并了包括应用程序官方描述和相似语言的评论回答的上下文知识, 提高了生成的评论回复的相关性和准确率.

(3) 跟踪链接发现 (traceability links discovery)

跟踪链接发现任务是通过不同的技术发现软件工件之间的关联关系, 如问题报告与修复问题报告的提交之间的链接. 准确的发现软件工件间的关联为后续挖掘软件开发过程有效信息, 挖掘漏洞等提供了有利保障, 部分研究者将预训练模型技术引入到跟踪链接任务发现中, 提升跟踪链接发现的性能. 如, Lin 等人[40]提出了结合 BERT 预训练模型的 T-BERT 方法来恢复开源项目中的问题和提交之间的链接, 产生比以前实现的精度高得多的跟踪链接.

3.2.2 性能分析评价

为了更全面地展示目前的预训练模型在 SE 领域自然语言任务中的性能, 表 2 中给出了以上自然语言相关任务常用预训练模型及其性能的统计与比较.

表 2 自然语言相关任务的性能比较和统计

NL任务	子任务	PTM	Accuracy	Precision	Recall	$F1$	MCC	AUC	BLEU	mAP
SE相关文本分类	问题报告分类	BERT-base	—	0.76	0.85	0.77	—	—	—	—
		BERToverflow	—	0.80	0.86	0.81	—	—	—	—
		seBERT	—	0.79	0.89	0.80	—	—	—	—
		RoBERTa	**0.82**	**0.84**	**0.86**	**0.85**	—	—	—	—
	APP评论分类	BERT	—	—	—	**0.94**	**0.61**	**0.79**	—	—
		RoBERTa	—	—	—	0.94	0.59	0.78	—	—
		XLNet	—	—	—	0.94	0.52	0.75	—	—
	文本情感分类	DeepMoji	—	**0.95**	**0.95**	**0.95**	—	—	—	—
		BERT	—	0.77	0.79	0.78	—	—	—	—
		RoBERTa	—	0.77	0.80	0.79	—	—	—	—
评论回复自动生成	—	RoBERTa	—	—	—	—	—	—	0.24	—
		BERT	—	—	—	—	—	—	**0.57**	—
跟踪链接发现	—	BERT	—	—	—	**0.97**	—	—	—	**0.99**

从表 2 可以得出以下结论.

• 软件工程相关文本分类中常用的性能评价指标主要有: Recall、*F*1、Accuracy、Precision、MCC、AUC、BLEU 和 mAP 等.

• 评论回复自动生成任务的常用性能评价指标主要有: BLEU 等.

• 跟踪链接发现任务的性能评价指标主要有: *F*1、mAP 等.

SE 领域自然语言任务中的以上性能指标中, 其值越大, 表明模型性能越好. 同时, 我们发现目前 SE 领域自然语言相关任务中, 直接使用自然语言处理中现有预训练模型的较多, 如 BERT、seBERT、RoBERTa、XLNet 等, 也有少量的基于 SE 特定领域的预训练模型, 如 BERToverflow. 但这些预训练模型在 SE 任务中并不能全部得到最高的性能. 可能这与自然语言任务主要是处理的软件开发领域文本数据有关, 而通用预训练模型的数据量要远远大于基于 SE 的特定领域模型. 同样地, 在 SE 自然语言相关任务中, 相同预训练模型在同一任务上的表现是不一样的, 如 BERT 在 APP 评论分类和评论回复自动生成任务中表现最好, 而在问题报告分类中反而 RoBERTa 表现更好. 因此, SE 领域研究者还需要进一步分析 SE 预训练模型的可解释性, 即产生的 Embedding 的差异性来更好地理解特定领域预训练模型和现有预训练模型的区别, 进而更好的指导 SE 自然语言相关任务.

3.3 程序语言与自然语言交互任务

3.3.1 任务描述

软件工程领域中存在着程序语言与自然语言间的交互, 如为了更好地理解代码, 会在代码中加入自然语言描述的注释. 同时, 在软件维护过程中, 审查人员会对开发人员提交的代码进行评审等. 因此, 为了帮助开发人员高效开发高质量软件, 软件工程领域研究者提出了各种方法解决程序语言与自然语言交互任务, 如代码摘要[32,44,66,74,75,77,78,81,82,85,89,92,94,97,99,100,106]、代码生成[32,41,42,44,55,56,75,88,89,95,96,101,115]、代码审查[64,69]、代码搜索[44,45,74,77,83,86,90,91,98–100]等任务.

(1) 代码摘要 (code summarization)

代码摘要的任务是为了帮助开发者更便捷地理解成程序语言代码, 依据上下文代码, 采用不同技术对代码生成自然语言的总结文本, 即 PL-To-NL 任务. 如代码注释生成、代码提交摘要生成等. 研究者已引入不同的预训练模型来深入挖掘程序语言与自然语言之间的关系, 进而产生高质量代码摘要. 如, Jung 等人[78]使用软件工程特定领域的 CodeBERT 预训练模型来挖掘程序语言与自然语言之间的巨大差异性, 成功提升了代码提交摘要生成任务的性能. Mastropaolo 等人[66]使用英语文本和程序源代码组成的数据集训练 text-to-text Transformer 网络, 形成了特定领域的 T5 预训练模型, 并将该模型应用于代码注释生成任务中, 提高了代码注释生成的性能. Khan 等人[94]将 Codex 应用于代码摘要生成任务中, 实验结果表明 Codex 提升代码摘要生成任务的性能. Liu 等人[106]使用 GitHub 上 7 种编程语言的 commits 训练编码器-解码器 Transformer 模型, 提出了 CommitBART 方法, 实验结果表明 CommitBART 优于以前的代码提交摘要生成方法.

(2) 代码生成 (code generation)

代码生成任务是与代码摘要完全相反的任务, 是从其自然语言描述中生成一个程序源代码 (在目标程序语言中) 的任务, 即 NL-To-PL 任务. Wang 等人[32]提出了一个统一的预训练编码-解码器的 Transformer 模型 CodeT5, 更好地利用了从开发人员分配的标识符中传递的代码语义, 并将 CodeT5 模型应用于 Java 语言代码生成中, 取得了较好的性能. Dong 等人[41]提出了基于 PDA (pushdown automation) 的 CodePAD 序列模型, 充分利用 PL 是 PDA 可识别语言的子集并且 PDA 接受的代码是符合语法的原则解决代码生成过程中的 PL 语法约束, 实验结果表明 CodePAD 可以利用基于序列的模型实现 100% 的语法正确率. Poesia 等人[42]针对目前大型预训练模型违反 PL 的句法和语义规则的问题, 提出了提高代码生成预训练模型 (GPT-3 和 Codex) 可靠性的 Synchromesh 框架, 其首先通过 TST (target similarity tuning) 从训练库中选择少量样本, 然后 Synchromesh 将这些样本提供给预训练模型 (如 GPT-3), 并使用约束语义解码器 (CSD) 对程序采样, 进而输出限定魏目标 PL 的有效代码.

(3) 代码审查 (code review)

代码审查任务是开源和工业项目中广泛采用保障软件质量的一种实践过程, 考虑到这一过程不可忽视的成

本, 研究者已研究通过预训练模型技术自动化特定代码审查任务. 可以分为代码更改质量评估, 根据提交审查的代码生成评审人评审 (即 PL-To-NL 任务) 和根据提交审查代码审查人评论生成评审人所要求的更改 (PL+NL-To-PL). 如 Tufano 等人[64]使用 Java 和英语组成的训练集预训练 T5 模型, 并将该模型应用于代码审查任务中, 实验结果发现预训练的 T5 模型可以得到优于之前 DL 模型的性能. Li 等人[69]从 9 种最流行编程语言的开源项目中收集真实代码更改和代码审查数据集, 并预训练 T5 模型得到 CodeReviewer 方法, 实验结果表明 CodeReviewer 的预训练任务和多语言预训练数据集有利于模型理解代码更改和审查.

(4) 代码搜索 (code search)

代码搜索任务是给定一种自然语言作为输入, 从一组程序源代码中找到语义上最相关的代码, 即 NL-To-PL 任务. Feng 等人[74]提出了特定领域的 CodeBERT 预训练模型, 并应用于代码搜索任务中, 发现 CodeBERT 预训练模型可以提升代码搜索任务的性能. 同样, Wang 等人[45]调研了不同的预训练模型 (例如, 现有模型 RoBERTa、特定领域模型 CodeBERT/GraphCodeBERT) 解决代码搜索任务的能力, 发现 GraphCodeBERT 预训练模型可以在代码搜索任务中得到最高的性能. Li 等人[83]通过大规模代码文本采用单模态对比和双模态对比训练来学习函数级代码语义表示, 提出 CodeRetriever 模型. 在单模态对比学习中, 基于文档和函数名称构建语义相关的代码对进行无监督学习. 在双峰对比学习中, 采用文档和代码的内联注释来构建代码文本对, 实验结果表明 CodeRetriever 显著改进了现有的代码预训练模型, 提高了代码搜索任务的性能.

3.3.2 性能分析评价

为了更全面理解预训练模型在程序语言与自然语言交互任务中的表现, 我们统计分析了程序语言与自然语言交互任务中常用的预训练模型, 并比较分析了这些预训练模型在其任务中的性能, 如表 3 所示.

表 3 程序语言与自然语言交互任务的性能比较和统计

NL与PL交互任务	PTM	Accuracy	Precision	Recall	*F*1	BLEU	CodeBLEU	EM	Pass@*k*	MRR	ROUGE	METEOR
代码摘要	GraphCodeBERT	—	—	—	—	0.46	—	—	—	—	0.57	0.27
	TreeBERT	—	**0.46**	**0.34**	**0.39**	0.48	—	—	—	—	0.57	0.27
	SPT-Code	—	—	—	—	**0.49**	—	—	—	—	**0.58**	**0.32**
代码生成	CommitBART	—	—	—	—	**0.59**	—	**0.30**	—	—	—	—
	StructCoder	—	—	—	—	0.41	0.45	0.22	—	—	—	—
	Codex	—	—	—	—	—	—	—	**0.29 (1)**, **0.47 (10)**, **0.72 (100)**	—	—	—
代码审查	T5	0.74	0.79	0.66	0.72	—	0.82	—	—	—	—	—
	CodeT5	0.67	0.70	0.59	0.64	—	—	—	—	—	—	—
代码搜索	UniXcoder	—	—	—	—	—	—	—	—	0.74	—	—
	SynCoBERT	—	—	—	—	—	—	—	—	0.74	—	—
	SCodeR	—	—	—	—	—	—	—	—	**0.77**	—	—

从表 3 中, 我们可以得出以下结论.

• 代码摘要生成任务常用的性能评价指标主要有: BLEU, ROUGE, METEOR 等. 其中, ROUGE 通过将模型生成的摘要与正确答案按 n-gram 拆分后, 计算召回率来衡量生成摘要与正确答案的匹配程度. METEOR 用来测量基于单精度的加权调和平均数和单字召回率, 其可以解决 BLEU 标准中单纯基于精度的问题. 同时与 BLEU 比较, 基于召回率的 ROUGE 和 METEOR 和人工判断的结果更相关. BLEU, ROUGE 和 METEOR 越大, 模型越好.

• 代码生成任务的性能评价指标主要有: BLEU, CodeBLEU, EM, 和 Pass@*k* (*k*=1, 10, 100) 等. 其中 Pass@*k* 是给定生成的 *n* (*n*≥*k*) 个样本, 计算通过单元测试的样本的数量, 并计算无偏估计量.

• 代码审查任务的性能评价指标主要有: Accuracy, Precision, Recall, *F*1, BLEU, EM 等. 其中, Accuracy, Precision, Recall, *F*1 用于评估代码更改质量, BLEU 和 EM 用于评价审查评论生成.

• 代码搜索任务的性能评价指标主要有: MRR (mean reciprocal rank) 等. MRR 用来评估搜索算法的统计量指

标, 为搜索 N 次代码结果倒数排名的平均值, 其中, 倒数排名为搜索 N 次的第一正确答案排名的倒数乘积, MRR 表明了搜到的代码是否摆在用户更明显的位置, MRR 越大, 模型越好.

同时, 我们发现程序语言与自然语言交互任务中, 常用的预训练模型是源代码预训练模型, 如 GraphCodeBERT, TreeBERT, Codex, SynCoBERT 等, 没有通用预训练模型及基于软工特定领域的预训练模型应用到该相关任务中. 同样地, 源代码预训练模型在不同任务中的表现是不一样的, 而这些源代码预训练模型由于使用的是不同的源代码表示, 以及不同的模型架构 (如 BERT, T5), 因此, 软件工程领域研究者需要进一步的理解源代码的多模态表示与模型架构的重要程度, 从而更好地指导交互任务.

3.4 软件工程领域图像相关任务

3.4.1 任务描述

目前软件工程领域图像相关任务中, UML 图分类任务[59,116]使用了预训练模型. 统一建模语言 (UML) 是一种公认的表示软件系统设计的标准建模语言, 可以帮助开发人员有效地研究、分析、编写文档、设计或开发任何软件. 对于学术研究, 需要包含 UML 图的大型案例, 收集此类数据集的挑战之一, 因此需要自动确定图像是否为 UML 图, 以及图像包含何种类型的 UML 图. 软件工程领域研究者提出了自动化 UML 图分类任务[116], 即对 UML 图类型分类为类图、活动图、序列图、用例图和非 UML 图等. UML 的软件工件为图片数据, 研究者已将图像处理领域的预训练模型应用到 UML 图分类任务中, 与传统的深度学习方法相比, 提升了 UML 图分类的性能. 如 Best 等人[59]将预训练在 ImageNet 的 VGG 模型引入 UML 图分类任务中, 通过少量的 UML 图数据集微调的预训练模式提升了 UML 图分类的准确率, 达到 90% 以上.

3.4.2 性能分析评价

我们统计分析了目前在软件工程领域图像相关任务中常用的预训练模型和性能, 发现目前主要是使用 VGG-16 预训练模型应用到 UML 图分类中, 其主要通过 Accuracy 来进行性能评价. 而在图形图像领域, 已经很更多更好地预训练模型被提出, 因此, 软件工程领域研究者需进一步地将图形图像领域的预训练模型应用到更多的软件工程任务中, 如基于 UI 的代码生成任务[61,62,117]等, 进一步提升图像相关任务中的自动化程度.

综上所述, 研究者已将预训练模型 (例如, 现有预训练模型和特定领域预训练模型) 广泛地应用于软件工程下游任务中, 特别是与程序语言和自然语言相关的任务中. 但研究发现特定领域预训练模型 (即, 训练在 SE 领域数据集的模型) 在 SE 领域任务中并一定优于现有预训练模型. 所以, 提出更符合 SE 领域特性的特定领域预训练模型是今后进一步的研究方向. 同时, 预训练模型重用技术在软件工程领域基于图像或声音的软件工件 (例如, UI 界面的自动生成任务[117], 基于语言的代码生成任务[118]等) 任务中的应用也是今后进一步的研究方向. 图 4 给出了目前软件工程领域下游任务的文献分布情况.

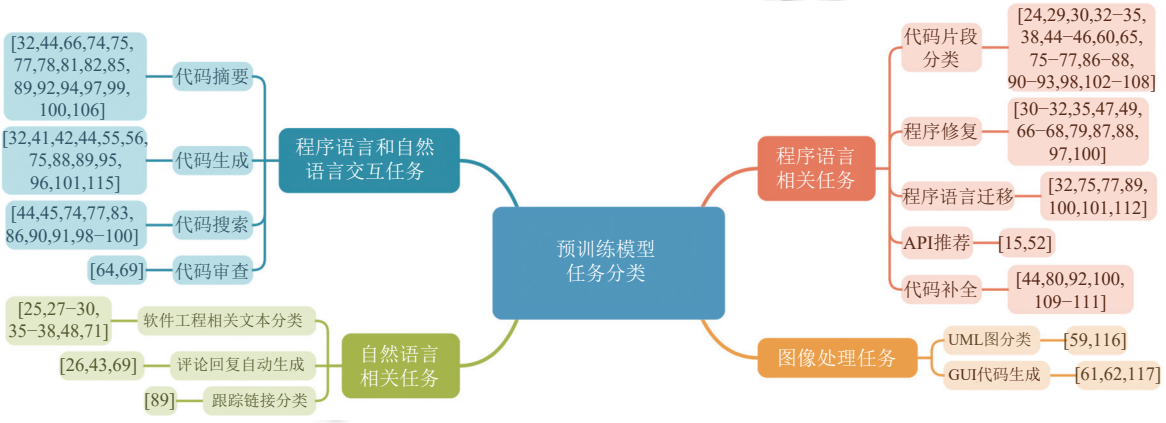


图 4 软件工程领域下游任务的文献分布

4 数据集

数据集是深度学习模型的基础, 其质量直接影响模型的性能. 特别是基于预训练模型的智能软件方法更离不开软件工程领域数据集, 如, 特定领域预训练模型或源代码预训练模型需要大量的软件工程领域数据集来训练模型, 完成下游任务需要少量软件工程下游任务数据来微调预训练模型. 为了更好地理解基于预训练模型的智能方法中使用到的数据集, 我们将分别从预训练模型数据集和下游任务数据集两方面进行总结和分析软件工程领域的数据集.

4.1 预训练模型数据集

自然语言文本与软件工程领域数据集有很大不同导致自然语言处理中预训练模型不能很好地适应软件工程任务. 为了更好地解决软件工程领域任务, 研究者收集大量的软件工程领域数据集从零训练深度学习模型, 提出了特定领域预训练模型和源代码预训练模型. 其中, 目前常用的大量软件工程领域数据集包括 SE 文本数据集、源代码数据集和 SE 文本和源代码的混合数据集. 表 4 给出了目前常用的预训练模型数据集. 从表 4 可以看出, 软件工程领域研究者主要从 Stack Overflow、GitHub 和 JIRA 来收集文本和源代码数据集训练软件工程领域预训练模型, 文本数据主要是以英语为主, 源代码数据主要是以 Java 和 Python 为主. 同时, 从表 4 中我们可以发现, 目前大部分的预训练模型都是针对一种程序语言的代码进行训练, 基于多语言源代码的预训练模型还较少. 另外, 预训练模型所基于的训练集的大小跨度也比较大, 从 0.53 MB 到 655 GB.

表 4 预训练模型数据集

类型	数据名称	语言	来源	数据量	公开时间	链接	使用的预训练模型
PL	CodeSearchNet+ C/C# datasets	Ruby/JavaScript/GO/ Python/Java/PHP/C/C#	GitHub+ BigQuery	8.35 GB	2021	https://github.com/salesforce/CodeT5	CodeT5[32,35,112]
	GitHub C language repositories	C	GitHub	5.8 GB	2020	—	C-BERT[33]
	Java and TypeScript datasets	Java/TypeScript	GitHub	—	2020	—	CugLM[80]
	Java datasets	Java	GitHub	—	2021	—	SynFix[49]
	CLCDSA dataset	Java/C/C++	AtCoder+ CodeJam	17.6 MB	2019	https://github.com/Kawser-nerd/CLCDSA/	IR-BERT[72]
	Java datasets	Java	GitHub	32 GB	2020	https://s3.amazonaws.com/code2vec/data/java14m_data.tar.gz	InferCode[90]/ code2vec[31,73]
	ETH Py150 Open corpus	Python	GitHub	190 MB	2020	https://github.com/google-research-datasets/eth_py150_open	CodeTrek[93]
	unique Python files	Python	GitHub	159 GB	2021	—	Codex[56]
	JavaSmall and JavaMed datasets	Java	GitHub	4.7 MB	2020	https://github.com/tech-srl/code2seq/tree/master/JavaExtractor and https://github.com/tech-srl/code2vec/tree/master/JavaExtractor	Coder[99]
	Python and Java pre-training corpus	Java/Python	GitHub	21.3 MB	2021	—	CuBERT[87]/ TreeBERT[85]
NL	SE textual data	English	Stack Overflow/ GitHub/Jira	119.7 GB	2021	https://github.com/smartshark/seBERT/tree/main/data	seBERT[29]
	CoNLL-2003	English	Stack Overflow	3.16 MB	2020	https://aclanthology.org/attachments/2020.acl-main.443.Dataset.zip	BERToverflow[29,38]/ CosSensBERT[92]

表 4 预训练模型数据集 (续)

类型	数据名称	语言	来源	数据量	公开时间	链接	使用的预训练模型
NL+PL	Java datasets from CodeSearchNet+SO posts	Java/English	GitHub+SO	52.5 MB	2022	https://zenodo.org/record/5387856#.Y1TB2nZByUl	T5[64]
	Java datasets from CodeSearchNet+SO posts	Java/English	GitHub	1.5 MB	2021	https://drive.google.com/drive/folders/1uJv-kljY1Q59fa-TdkpXOOd9QEG5OZDa?usp=sharing	T5[66]
	Java and Python from BigQuery+SO posts	Java/Python/English	BigQuery+GitHub	655 GB	2021	https://github.com/wasiahmad/PLBART	PLBART[75]
	CodeSearchNet	Ruby/JavaScript/GO/Python/Java/PHP/English	GitHub	3.5 GB	2019	https://github.com/github/CodeSearchNet	T-BERT[107]/GraphCodeBERT[84]/CodeBERT[74,51]
	AnghaBench	C	GitHub	0.53 MB	2020	http://cuda.dcc.ufmg.br/angha/home	COMBO[65]

4.2 下游任务数据集

为了更好地完成软件工程领域的下游任务, 研究者需要少量的下游任务数据集来微调基于预训练模型的智能方法. 本节整理了目前常用的下游任务数据集, 根据任务的不同, 下游任务数据集可以分为 SE 文本数据集 (NL)、源代码数据集 (PL)、源代码和文本混合数据集以及图片数据集等. 表 5 给出了目前常用的下游任务数据集. 从表 5 可以看出, 目前针对 SE 的各任务已有丰富的公开的数据集供研究者使用, 在 NL 和 PL 交互任务中主要以 Python 语言位置, 在 PL 的代码片段分类任务中, 特别是漏洞检测任务中主要是以 C/C++语言为主, 而在程序修复任务中, 以 Java 和 JavaScript 程序语言为主, SE 领域的研究者可以收集更丰富语言的数据集. 同时, 研究者可以收集更多的图像领域的数据集, 使用 CV 领域的预训练模型更好地改进 SE 的任务.

表 5 下游任务数据集

类型	任务	数据名称	相关文献	语言	样本量	公开时间	链接
PL	代码片段分类	Java patches	[31]	Java	102 041	2020	https://github.com/TruX-DTF/DL4PatchCorrectness
		POJ104	[45,60,76,77]	C/C++	30 815	2021	—
		CodeCloneBench	[45,77,107]	Java	901 028	2014	—
		SATD dataset	[30]	—	—	2016	https://github.com/maldonado/tse.satd.data/tree/master/dataset
		SmartBugs wild dataset	[103]	—	47 398	2020	https://github.com/smartbugs/smartbugs-wild
		SPI	[106]	C	298 917	2021	https://sites.google.com/view/du-commits/home
		QEMU	[105]	C/C++	13 600	2005	https://www.qemu.org/index.html
		FFmpeg	[105]	C/C++	4 919	2006	https://www.linuxjournal.com/article/8517
		Devign	[91,102]	C	27 318	2021	https://github.com/microsoft/CodeXGLUE/tree/main/Code-Code/Defect-detection
		Merge conflicts dataset	[108]	C#/JavaScript/TypeScript/Java	219 934	2022	—
		Multi-language commit message dataset (MCMD)	[94]	Java/C#/C++/Python/JavaScript	2 250 000	2022	https://anonymous.4open.science/r/CommitMessageEmpirical/README.mdd
		Vuldeepecker	[46]	C/C++	61 638	2018	https://github.com/CGCL-codes/VulDeePecker
		Draper	[46]	C/C++	1 274 366	2018	https://osf.io/d45bw/
		REVEAL	[46]	C/C++	18 169	2020	https://github.com/VulDetProject/ReVeal/tree/master/data

表 5 下游任务数据集 (续)

类型	任务	数据名称	相关文献	语言	样本量	公开时间	链接
PL	代码片段分类	muVuldeepecker (MVD)	[46]	C/C++	181 641	2019	https://github.com/muVulDeePecker/muVulDeePecker
		D2A	[46]	C/C++	1 295 623	2021	https://github.com/ibm/D2A
	程序修复	ManySStuBs4J	[79]	Java	63 923	2021	https://github.com/EhsanMashhadi/MSR2021-ProgramRepair/tree/main/data
		Automatic bug fixing	[66]	Java	46 680	2019	https://drive.google.com/drive/folders/1uJv-kljY1Q59fa-TdkpXOOd9QEG5OZDa?usp=sharing
		TFix-dataset	[67]	JavaScript	104 804	2021	https://drive.google.com/file/d/1CtfnYaVf-q6FZP5CUM4Wh7ofpp8b9ajW/view
		QuixBugs	[68,97]	Python/Java	—	2017	https://github.com/jkoppel/QuixBugs
		CoCoNut	[68]	Java/Python/C/JavaScript	9 675 342	2020	https://github.com/lin-tan/CoCoNut-Artifact/releases
		BugAID	[68]	JavaScript	—	2016	http://salt.ece.ubc.ca/software/bugaid/
		ManyBugs	[68]	C	10 468	2015	https://repairbenchmarks.cs.umass.edu/ManyBugs/
	代码补全	Java and TypeScript datasets	[80]	Java/TypeScript	—	2020	—
		ETH Py150 corpus	[44,85,86,88,92,93]	Python	74 749	2020	https://github.com/google-research-datasets/eth_py150_open
	API 推荐	Req2Lib-dataset	[52]	Java	5 625	2020	—
	程序语言迁移	CodeTrans	[40]	Java/C#	10 300	2021	https://github.com/microsoft/CodeXGLUE
		Python800 dataset	[86]	Python	240 000	2021	https://github.com/IBM/Project_CodeNet
NL	文本分类	Herzig's issue report datasets	[29,36]	English	—	2012	http://www.st.cs.uni-saarland.de/softevo/bugclassify/
		Commit messages	[24,78]	English	1 793	2021	https://zenodo.org/record/4266643#.X6vERuLPxPY
		issue report from GitHub	[48]	English	—	2021	https://bitbucket.org/jstzwj/lms4githubissue/src/master/data_view.7z
		SEntiMoji dataset	[71]	—	10 096	2019	https://github.com/SEntiMoji/SEntiMoji
	评论回复自动生成	review-response pairs datasets	[26,43]	English	570 881	2020	—
	跟踪链接发现	traceability dataset	[40]	English	1834	2021	https://zenodo.org/record/4511291#.YB3tjyj0mbg
PL+NL	代码摘要	Code review comments (CR)	[29,36]	—	1 600	2017	https://github.com/senticr/SentiCR/
		Code summarization (CS)	[66]	Java	1 953 940	2020	https://drive.google.com/drive/folders/1uJv-kljY1Q59fa-TdkpXOOd9QEG5OZDa?usp=sharing
		CodeSearch-Net	[44,70,83,88,94,98,100,101,108]	Ruby/JavaScript/GO/Python/Java/PHP	—	2019	https://github.com/github/CodeSearchNet
	代码生成	Concode data	[32,40,44,55,89,101]	Java	100 000	2018	https://drive.google.com/drive/folders/1kC6fe7JgOmEHhVFaXjzOmKeatTJy1I1W
		DJANGO	[41]	Python	18 805	2015	—
		JUICE-10K	[41]	Python	13 946	2019	https://github.com/rajasagashe/juice
		MBPP	[41,95,115]	Python	974	2021	https://github.com/google-research/google-research/tree/master/mbpp
		Spider	[42,96]	SQL	5 693	2018	https://yale-lily.github.io/spider
		APPS	[56,95]	Python	232 421	2021	https://github.com/hendrycks/apps
		CodeContests	[95]	C++/Python/Java	13 610	2022	https://github.com/deepmind/code_contests

表 5 下游任务数据集 (续)

类型	任务	数据名称	相关文献	语言	样本量	公开时间	链接
PL+NL	综合	HumanEval	[56,95,115]	Python	—	2021	https://github.com/openai/human-eval
		CodeXGLUE	[44,75,82,110]	—	—	2021	https://github.com/microsoft/CodeXGLUE
	代码搜索	AdvTest dataset	[44,83,86,91,98]	Python	280 634	2021	https://github.com/microsoft/CodeXGLUE/tree/main/Text-Code/NL-code-search-Adv
		CoNaLa	[83,86,95]	Python/Java	79 809	2018	https://conala-corpus.github.io/
		SO-DS	[83]	Python	13 250	2020	https://github.com/nokia/codesearch
		StaQC	[83]	Python	147 546	2018	https://github.com/LittleYUYU/StackOverflow-Question-Code-Dataset
		CoSQA	[44,83,86]	Python	20 604	2021	https://github.com/Jun-jie-Huang/CoCLR/tree/main/data
CV	UML图片分类	UML图像	[33]	—	14 815	2016	—

注: 相关文献列给出了使用对应数据集的文献

综上所述, 软件工程领域研究者已经提供了较丰富的数据集来训练或者微调下游任务, 并且大部分数据集已对外开源, 为后续的研究者提供更好的预训练模型提供便利. 但是, 从公开的数据集可以看出, 文本数据以英语为主, 预训练模型和软件工程下游任务的源代码数据集基于的程序设计语言都相对单一. 因此, 为了能提出泛化地更好适应各种程序语言或者自然语言的模型, 软件工程领域研究者还需要进一步提供更加丰富的数据集.

5 总结与展望

BERT 和 VGG 等预训练模型在自然语言处理和图像处理中的成功应用受到了软件工程领域研究者的广泛关注, 研究者们将预训练模型技术应用于软件工程任务中, 缓解了数据集带来的挑战, 对处理软件工程下游任务提供了新的方式. 本文围绕研究框架、常用的预训练模型、预训练范式、软件工程下游任务和常用软件工程领域数据集系统梳理了 2015 年以来基于预训练模型的智能软件工程方法相关工作. 从总结分析的文献中可以看出, 大部分的工作都集中利用软件工程领域文本和程序语言源代码数据集, 使用现有的自然语言预训练模型和基于源代码训练的源代码模型应用于软件工程领域源代码、文本以及源代码和文本交互任务, 并取得了一定的成果. 但由于软件工程领域文本与程序语言数据的复杂性, 基于预训练模型的智能软件工程方法挑战仍然存在, 存在着大量值得进一步研究的问题.

(1) 新 PTM 在软件工程领域的应用值得关注

尽管基于预训练模型的方法在 SE 相关任务中的应用是明显的, 但目前软件工程领域研究者大多聚焦在使用自然语言处理的 BERT 及其变体 (例如, RoBERTa) 模型来解决软件工程领域代码片段分类、代码生成、代码摘要等问题. 而 Watson 等人[1]调研发现不同的 DL 框架适合处理特定的不同 SE 任务. 换句话说, 目前软件工程领域研究者只是使用少量的预训练模型技术解决少量软件工程任务. 因此, 在软件工程领域一个值得关注的研究点是将新 PTM 模型应用于普遍探索的 SE 任务中.

一方面需要进一步引入深度学习领域内预训练模型的最新研究成果, 更好的编码表示 SE 领域工件, 如软件工程领域自然语言及不同程序语言的语法语义信息特征, 进而提升软件工程领域下游任务的性能.

另一方面探索软件工程领域下游任务与不同预训练模型的关联, 对于不同 SE 下游任务给出选择合适的预训练模型技术的指导, 比如代码生成, 代码摘要和代码搜索任务尽管都是 NL 和 PL 交互任务, 但是由于各自的任务特征, 相同的预训练模型在这些任务中的表现是不同的, 研究者应该根据任务特征探究如何选择预训练模型来更好的选择预训练模型.

(2) 组合多数据类型的预训练数据集值得关注

目前, 软件工程领域基于预训练模型的智能方法研究中绝大多数使用多个项目或多领域的源代码和文本数据作为训练集, 在编程语言上大多数只选择至多 2 种程序语言构建预训练模型. 研究中指明使用的有限数据类型训

练集不具有代表性, 限制了模型性能的泛化性. 比如使用 Java 语言训练的预训练模型无法很好地对 Python 或 C 语言代码知识表示, 需要针对不同语言构建不同预训练模型. 因此, 建立软件工程领域的特定预训练模型, 除了分析一些被充分代表的数据类型外, 有机组合多种数据类型 (即多模态及跨语言数据集) 值得关注, 从而训练出泛化的预训练模型. 如对于源代码的 SE 任务, 除了考虑源代码的文本描述, 还可以组合代码注释、源代码的抽象语法树 (AST) 和数据流图 (DFD) 表示及相同语义的其他程序语言代码形成多模态跨语言的源代码数据集.

另外, 目前软件工程领域基于图像数据集解决下游任务还较少, 如只有使用 UML 图像数据训练图像处理预训练模型解决 UML 图分类任务. 因此, 研究者可以收集更多更高质量的图像数据实现更多的软件工程领域下游任务, 如根据数据流图或类图实现代码的自动生成任务.

(3) 如何依据下游任务数据实现预训练模型微调值得关注

目前软件工程领域基于预训练模型的智能方法主要采用预训练-特征表示和预训练-微调两种预训练范式. 其中, 微调是将 PTM 应用到软件工程下游任务中的主要方法. 但由于软件工程领域数据存在本地性[119,120], 使得每个软件工程领域下游任务都要根据其数据进行参数微调, 导致参数效率低下问题. 而且若当下游任务仅有少量数据时, 微调预训练模型会导致模型过拟合和欠拟合微调数据集情况. 因此, 如何弥合预训练和特定任务微调之间的差距值得关注.

一方面可以固定 PTM 的原始参数, 并通过为特定任务添加小的可微调的自适应模块[91]. 这样, 可以使用一个共享的 PTM 来服务多个软件工程领域下游任务, 更加灵活挖掘 PTM 中的知识.

另一方面考虑不从头开始微调面向任务的预训练模型, 而是通过使用模型压缩技术从现有的 PTM 中提取部分特定任务的知识来学习, 因此研究者需要探索适合软件工程领域任务的模型压缩技术.

(4) 软件工程领域预训练模型的可解释性值得关注

尽管 PTM 在软件工程领域任务中达到了令人印象深刻的性能, 但其深层的非线性架构使得决策过程高度不透明. 而目前软件工程领域任务中绝大多数使用基于 Transformer 架构的 BERT 模型, 解释 PTM4SE 变得更困难. 因此, 作为解决软件工程领域下游任务的关键组件, 预训练模型在软件工程领域的可解释性需进一步探索, 这有助于我们理解 PTM 如何工作, 并更好地使用和进一步改进提供指导. 如研究者可以探究预训练模型是如何有效地学习软件工程领域自然语言文本和不同程序语言代码的语法语义信息, 预训练模型的不同模块 (如注意力机制、掩码语言建模) 在软件工程领域知识表示中扮演着怎样角色. 另外, 目前深度学习领域研究者设计了广泛的 Probing tasks 和 Visualization 来调查 PTM 的可解释性, 软件工程领域研究者可以考虑引入深度学习领域的可解释性来提升软件工程领域预训练模型的可解释性. 从而对软件工程领域的研究者提供更好的指引.

References:

[1] Watson C, Cooper N, Palacio DN, Moran K, Poshyvanyk D. A systematic literature review on the use of deep learning in software engineering research. ACM Trans. on Software Engineering and Methodology, 2022, 31(2): 32. [doi: 10.1145/3485275]

[2] Hellendoorn VJ, Devanbu P. Are deep neural networks the best choice for modeling source code? In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: Association for Computing Machinery, 2017. 763–773. [doi: 10.1145/3106237.3106290]

[3] Budhiraja A, Dutta K, Reddy R, Shrivastava M. DWEN: Deep word embedding network for duplicate bug report detection in software repositories. In: Proc. of the 40th Int'l Conf. on Software Engineering. Gothenburg: Association for Computing Machinery, 2018. 193–194. [doi: 10.1145/3183440.3195092]

[4] Liu BC, Huo W, Zhang C, Li WC, Li F, Piao A, Zou W. αDiff: Cross-version binary code similarity detection with DNN. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: Association for Computing Machinery, 2018. 667–678. [doi: 10.1145/3238147.3238199]

[5] Qiu XP, Sun TX, Xu YG, Shao YF, Dai N, Huang XJ. Pre-trained models for natural language processing: A survey. Science China Technological Sciences, 2020, 63(10): 1872–1897. [doi: 10.1007/s11431-020-1647-3]

[6] Niu CA, Li CY, Luo B, Ng V. Deep learning meets software engineering: A survey on pre-trained models of source code. In: Proc. of the 31st Int'l Joint Conf. on Artificial Intelligence. Vienna: IJCAI, 2022. 5546–5555. [doi: 10.24963/ijcai.2022/775]

[7] Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. Experimentation in Software Engineering. Berlin: Springer, 2012. [doi: 10.1007/978-3-642-29044-2]

[8] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: ACL, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[9] Radford A, Narasimhan K, Salimans T, Sutskever I. Improving language understanding by generative pre-training. 2018. https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf

[10] Yang ZL, Dai ZH, Yang YM, Carbonell J, Salakhutdinov R, Le QV. XLNet: Generalized autoregressive pretraining for language understanding. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 517.

[11] He KM, Zhang XY, Ren SQ, Sun J. Deep residual learning for image recognition. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). Las Vegas: IEEE, 2016. 770–778. [doi: 10.1109/CVPR.2016.90]

[12] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556, 2015.

[13] Liu Y, Liu MW, Peng X, Treude C, Xing ZC, Zhang XX. Generating concept based API element comparison using a knowledge graph. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2020. 834–845.

[14] Al Omran FNA, Treude C. Choosing an NLP library for analyzing software documentation: A systematic literature review and a series of experiments. In: Proc. of the 14th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Buenos Aires: IEEE, 2017. 187–197. [doi: 10.1109/MSR.2017.42]

[15] Xie WK, Peng X, Liu MW, Treude C, Xing ZC, Zhang XX, Zhao WY. API method recommendation via explicit matching of functionality verb phrases. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. New York: Association for Computing Machinery, 2020. 1015–1026. [doi: 10.1145/3368089.3409731]

[16] Zhao XJ, Xing ZC, Kabir MA, Sawada N, Li J, Lin SW. HDSKG: Harvesting domain specific knowledge graph from content of webpages. In: Proc. of the 24th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Klagenfurt: IEEE, 2017. 56–67. [doi: 10.1109/SANER.2017.7884609]

[17] Tian Y, Lo D. A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports. In: Proc. of the 22nd Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). Montreal: IEEE, 2015. 570–574. [doi: 10.1109/SANER.2015.7081879]

[18] Wu Q, Liang GT, Wang QX, Xie T, Mei H. Iterative mining of resource-releasing specifications. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2011). Lawrence: IEEE, 2011. 233–242. [doi: 10.1109/ASE.2011.6100058]

[19] Pandita R, Xiao XS, Zhong H, Xie T, Oney S, Paradkar A. Inferring method specifications from natural language API descriptions. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). Zurich: IEEE, 2012. 815–825. [doi: 10.1109/ICSE.2012.6227137]

[20] Howard MJ, Gupta S, Pollock L, Vijay-Shanker K. Automatically mining software-based, semantically-similar words from comment-code mappings. In: Proc. of the 10th Working Conf. on Mining Software Repositories (MSR). San Francisco: IEEE, 2013. 377–386. [doi: 10.1109/MSR.2013.6624052]

[21] Pandita R, Xiao XS, Yang W, Enck W, Xie T. WHYPER: Towards automating risk assessment of mobile applications. In: Proc. of the 22nd USENIX Conf. on Security. Washington: USENIX Association, 2013. 527–542.

[22] Wang C, Peng X, Liu MW, Xing ZC, Bai XF, Xie B, Wang T. A learning-based approach for automatic construction of domain glossary from source code and documentation. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: Association for Computing Machinery, 2019. 97–108. [doi: 10.1145/3338906.3338963]

[23] Liu YC, Sun XB, Duan YC. Analyzing program readability based on WordNet. In: Proc. of the 19th Int'l Conf. on Evaluation and Assessment in Software Engineering. New York: Association for Computing Machinery, 2015. 27. [doi: 10.1145/2745802.2745837]

[24] Ghadhab L, Jenhani I, Mkaouer MW, Messaoud MB. Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model. Information and Software Technology, 2021, 135: 106566. [doi: 10.1016/j.infsof.2021.106566]

[25] Biswas E, Karabulut ME, Pollock L, Vijay-Shanker K. Achieving reliable sentiment analysis in the software engineering domain using BERT. In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Adelaide: IEEE, 2020. 162–173. [doi: 10.1109/ICSME46990.2020.00025]

[26] Gao CY, Zhou WJ, Xia X, Lo D, Xie Q, Lyu MR. Automating APP review response generation based on contextual knowledge. ACM Trans. on Software Engineering and Methodology, 2022, 31(1): 11. [doi: 10.1145/3464969]

[27] Wang J, Zhang XF, Chen L. How well do pre-trained contextual language representations recommend labels for GitHub issues?

Knowledge-based Systems, 2021, 232: 107476. [doi: 10.1016/j.knosys.2021.107476]

[28] Hadi MA, Fard FH. Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of APP-reviews. arXiv:2104.05861, 2021.

[29] von der Mosel J, Trautsch A, Herbold S. On the validity of pre-trained Transformers for natural language processing in the software engineering domain. IEEE Trans. on Software Engineering, 2023, 49(4): 1487–1507. [doi: 10.1109/TSE.2022.3178469]

[30] Prenner JAA, Robbes R. Making the most of small software engineering datasets with modern machine learning. IEEE Trans. on Software Engineering, 2022, 48(12): 5050–5067. [doi: 10.1109/TSE.2021.3135465]

[31] Tian HY, Liu K, Kaboré AK, Koyuncu A, Li L, Klein J, Bissyandé TF. Evaluating representation learning of code changes for predicting patch correctness in program repair. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: Association for Computing Machinery, 2020. 981–992. [doi: 10.1145/3324884.3416532]

[32] Wang Y, Wang WS, Joty S, Hoi SCH. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proc. of the 2021 Conf. on Empirical Methods in Natural Language Processing. Punta Cana: ACL, 2021. 8696–8708. [doi: 10.18653/v1/2021.emnlp-main.685]

[33] Buratti L, Pujar S, Bornea M, McCarley S, Zheng YH, Rossiello G, Morari A, Laredo J, Thost V, Zhuang YF, Domeniconi G. Exploring software naturalness through neural language models. arXiv:2006.12641, 2020.

[34] De Bortoli Fávero EM, Casanova D, Pimentel AR. SE^{3}M: A model for software effort estimation using pre-trained embedding models. Information and Software Technology, 2022, 147: 106886. [doi: 10.1016/j.infsof.2022.106886]

[35] Karmakar A, Robbes R. What do pre-trained code models know about code? In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 1332–1336. [doi: 10.1109/ASE51524.2021.9678927]

[36] Zhang T, Xu BW, Thung F, Haryono SA, Lo D, Jiang LX. Sentiment analysis for software engineering: How far can pre-trained Transformer models go? In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Adelaide: IEEE, 2020. 70–80. [doi: 10.1109/ICSME46990.2020.00017]

[37] Houlsby N, Giurgiu A, Jastrzebski S, Morrone B, De Laroussilhe Q, Gesmundo A, Attariyan M, Gelly S. Parameter-efficient transfer learning for NLP. In: Proc. of the 36th Int'l Conf. on Machine Learning. Los Angeles: PMLR, 2019. 2790–2799.

[38] Yang CR, Xu BW, Khan JY, Uddin G, Han D, Yang Z, Lo D. Aspect-based API review classification: How far can pre-trained Transformer model go? In: Proc. of the 2022 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Honolulu: IEEE, 2022. 385–395. [doi: 10.1109/SANER53432.2022.00054]

[39] Uddin G, Guéhénuc YG, Khomh F, Roy CK. An empirical study of the effectiveness of an ensemble of stand-alone sentiment detection tools for software engineering datasets. ACM Trans. on Software Engineering and Methodology, 2022, 31(3): 48. [doi: 10.1145/3491211]

[40] Lin JF, Liu YL, Zeng QK, Jiang M, Cleland-Huang J. Traceability transformed: Generating more accurate links with pre-trained BERT models. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 324–335. [doi: 10.1109/ICSE43902.2021.00040]

[41] Dong YH, Jiang X, Liu YC, Li G, Jin Z. CodePAD: Sequence-based code generation with pushdown automaton. arXiv:2211.00818, 2023.

[42] Poesia G, Polozov A, Le V, Tiwari A, Soares G, Meek C, Gulwani S. Synchromesh: Reliable code generation from pre-trained language models. In: Proc. of the 10th Int'l Conf. on Learning Representations. ICLR, 2022.

[43] Cao Y, Fard FH. Pre-trained neural language models for automatic mobile APP user feedback answer generation. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering Workshops (ASEW). Melbourne: IEEE, 2021. 120–125. [doi: 10.1109/ASEW52652.2021.00033]

[44] Guo DY, Lu S, Duan N, Wang YL, Zhou M, Yin J. UniXcoder: Unified cross-modal pre-training for code representation. In: Proc. of the 60th Annual Meeting of the Association for Computational Linguistics. Dublin: Association for Computational Linguistics, 2022. 7212–7225. [doi: 10.18653/v1/2022.acl-long.499]

[45] Wang DZ, Jia ZY, Li SS, Yu Y, Xiong Y, Dong W, Liao XK. Bridging pre-trained models and downstream tasks for source code understanding. In: Proc. of the 44th Int'l Conf. on Software Engineering (ICSE). Pittsburgh: IEEE, 2022. 287–298. [doi: 10.1145/3510003.3510062]

[46] Hanif H, Maffeis S. VulBERTa: Simplified source code pre-training for vulnerability detection. In: Proc. of the 2022 Int'l Joint Conf. on Neural Networks (IJCNN). Padua: IEEE, 2022. 1–8. [doi: 10.1109/IJCNN55064.2022.9892280]

[47] Ahmed T, Ledesma NR, Devanbu P. SynShine: Improved fixing of syntax errors. IEEE Trans. on Software Engineering, 2023, 49(4): 2169–2181. [doi: 10.1109/TSE.2022.3212635]

[48] Izadi M, Akbari K, Heydarnoori A. Predicting the objective and priority of issue reports in software repositories. Empirical Software Engineering, 2022, 27(2): 50. [doi: 10.1007/s10664-021-10085-3]

[49] Ahmed T, Ledesma NR, Devanbu P. SynFix: Automatically fixing syntax errors using compiler diagnostics. arXiv:2104.14671, 2021.

[50] Efstathiou V, Chatzilenas C, Spinellis D. Word embeddings for the software engineering domain. In: Proc. of the 15th Int'l Conf. on Mining Software Repositories. Gothenburg: Association for Computing Machinery, 2018. 38–41. [doi: 10.1145/3196398.3196448]

[51] Pradel M, Murali V, Qian R, Machalica M, Meijer E, Chandra S. Scaffle: Bug localization on millions of files. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: Association for Computing Machinery, 2020. 225–236. [doi: 10.1145/3395363.3397356]

[52] Sun ZS, Liu Y, Cheng ZM, Yang C, Che PY. Req2Lib: A semantic neural model for software library recommendation. In: Proc. of the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). London: IEEE, 2020. 542–546. [doi: 10.1109/SANER48275.2020.9054865]

[53] Zhong H, Su ZD. Detecting API documentation errors. In: Proc. of the 2013 ACM SIGPLAN Int'l Conf. on Object Oriented Programming Systems Languages & Applications. Indianapolis: Association for Computing Machinery, 2013. 803–816. [doi: 10.1145/2509136.2509523]

[54] Liu MW, Peng X, Marcus A, Xing ZC, Xie WK, Xing SS, Liu Y. Generating query-specific class API summaries. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Tallinn: Association for Computing Machinery, 2019. 120–130. [doi: 10.1145/3338906.3338971]

[55] Perez L, Ottens L, Viswanathan S. Automatic code generation using pre-trained language models. arXiv:2102.10535, 2021.

[56] Chen M, Tworek J, Jun H, *et al*. Evaluating large language models trained on code. arXiv:2107.03374, 2021.

[57] Zhong H, Zhang L, Xie T, Mei H. Inferring resource specifications from natural language API documentation. In: Proc. of the 2009 IEEE/ACM Int'l Conf. on Automated Software Engineering. Auckland: IEEE, 2009. 307–318. [doi: 10.1109/ASE.2009.94]

[58] Ghanem M, Elnaggar A, Mckinnon A, Debes C, Boisard O, Matthes F. Automated employee objective matching using pre-trained word embeddings. In: Proc. of the 25th IEEE Int'l Enterprise Distributed Object Computing Conf. (EDOC). Gold Coast: IEEE, 2021. 51–60. [doi: 10.1109/EDOC52215.2021.00016]

[59] Best N, Ott J, Linstead EJ. Exploring the efficacy of transfer learning in mining image-based software artifacts. Journal of Big Data, 2020, 7(1): 59. [doi: 10.1186/s40537-020-00335-4]

[60] Keller P, Kaboré AK, Plein L, Klein J, Traon YL, Bissyandé TF. What you see is what it means! Semantic representation learning of code based on visualization and transfer learning. ACM Trans. on Software Engineering and Methodology, 2021, 31(2): 31. [doi: 10.1145/3485135]

[61] Soselia D, Saifullah K, Zhou TY. Reinforcement learning finetuned vision-code Transformer for UI-to-code generation. arXiv:2305.14637, 2023.

[62] Li G, Li Y. Spotlight: Mobile UI understanding using vision-language models with a focus. In: Proc. of the 11th Int'l Conf. on Learning Representations. Kigali: OpenReview.net, 2023. 1–16.

[63] Deng J, Dong W, Socher R, Li LJ, Li K, Li FF. ImageNet: A large-scale hierarchical image database. In: Proc. of the 2009 IEEE Conf. on Computer Vision and Pattern Recognition. Miami: IEEE, 2009. 248–255. [doi: 10.1109/CVPR.2009.5206848]

[64] Tufano R, Masiero S, Mastropaolo A, Pascarella L, Poshyvanyk D, Bavota G. Using pre-trained models to boost code review automation. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: Association for Computing Machinery, 2022. 2291–2302. [doi: 10.1145/3510003.3510621]

[65] Zhang YF, Huang C, Zhang YK, Cao K, Andersen ST, Shao HJ, Leach K, Huang Y. COMBO: Pre-training representations of binary code using contrastive learning. arXiv:2210.05102, 2022.

[66] Mastropaolo A, Scalabrino S, Cooper N, Nader Palacio D, Poshyvanyk D, Oliveto R, Bavota G. Studying the usage of text-to-text transfer Transformer to support code-related tasks. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 336–347. [doi: 10.1109/ICSE43902.2021.00041]

[67] Berabi B, He JX, Raychev V, Vechev M. TFix: Learning to fix coding errors with a text-to-text Transformer. In: Proc. of the 38th Int'l Conf. on Machine Learning. Vienna: PMLR, 2021. 780–791.

[68] Yuan W, Zhang QJ, He TK, Fang CR, Hung NQV, Hao XD, Yin HZ. CIRCLE: Continual repair across programming languages. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: Association for Computing Machinery, 2022. 678–690. [doi: 10.1145/3533767.3534219]

[69] Li ZY, Lu S, Guo DY, Duan N, Jannu S, Jenks G, Majumder D, Green J, Svyatkovskiy A, Fu SY, Sundaresan N. Automating code review activities by large-scale pre-training. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and the Symp. on

the Foundations of Software Engineering. Singapore: Association for Computing Machinery, 2022. 1035–1047. [doi: 10.1145/3540250.3549081]

[70] Mastropaolo A, Aghajani E, Pascarella L, Bavota G. An empirical study on code comment completion. In: Proc. of the 2021 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Luxembourg: IEEE, 2021. 159–170. [doi: 10.1109/ICSME52107.2021.00021]

[71] Chen ZP, Cao YB, Lu X Mei QZ, Liu XZ. SEntiMoji: An emoji-powered learning approach for sentiment analysis in software engineering. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Tallinn: Association for Computing Machinery, 2019. 841–852. [doi: 10.1145/3338906.3338977]

[72] Gui Y, Wan Y, Zhang HY, Huang HF, Sui YL, Xu GD, Shao ZY, Jin H. Cross-language binary-source code matching with intermediate representations. In: Proc. of the 2022 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Honolulu: IEEE, 2022. 601–612. [doi: 10.1109/SANER53432.2022.00077]

[73] Alon U, Zilberstein M, Levy O, Yahav E. code2vec: Learning distributed representations of code. Proc. of the ACM on Programming Languages, 2019, 3(POPL): 40. [doi: 10.1145/3290353]

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

[75] Ahmad W, Chakraborty S, Ray B, Chang KW. Unified pre-training for program understanding and generation. In: Proc. of the 2021 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. ACL, 2021. 2655–2668. [doi: 10.18653/v1/2021.naacl-main.211]

[76] Peng DL, Zheng SX, Li YT, Ke GL, He D, Liu TY. How could neural networks understand programs? In: Proc. of the 38th Int'l Conf. on Machine Learning. Vienna: PMLR, 2021. 8476–8486.

[77] Lachaux MA, Rozière B, Szafraniec M, Lample G. DOBF: A deobfuscation pre-training objective for programming languages. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. NeurIPS, 2021. 14967–14979.

[78] Jung TH. CommitBERT: Commit message generation using pre-trained programming language model. In: Proc. of the 1st Workshop on Natural Language Processing for Programming. ACL, 2021. 26–33. [doi: 10.18653/v1/2021.nlp4prog-1.3]

[79] Mashhadi E, Hemmati H. Applying CodeBERT for automated program repair of Java simple bugs. In: Proc. of the 18th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Madrid: IEEE, 2021. 505–509. [doi: 10.1109/MSR52588.2021.00063]

[80] Liu F, Li G, Zhao YF, Jin Z. Multi-task learning based pre-trained language model for code completion. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: Association for Computing Machinery, 2020. 473–485. [doi: 10.1145/3324884.3416591]

[81] Tao W, Wang YL, Shi ES, Du L, Han S, Zhang HY, Zhang DM, Zhang WQ. A large-scale empirical study of commit message generation: Models, datasets and evaluation. Empirical Software Engineering, 2022, 27(7): 198. [doi: 10.1007/s10664-022-10219-1]

[82] Chen N, Sun QS, Zhu RY, Li X, Lu XS, Gao M. CAT-probing: A metric-based approach to interpret how pre-trained models for programming language attend code structure. In: Proc. of the 2022 Findings of the Association for Computational Linguistics. Abu Dhabi: ACL, 2022. 4000–4008. [doi: 10.18653/v1/2022.findings-emnlp.295]

[83] Li XN, Gong YY, Shen YL, Qiu XP, Zhang H, Yao BL, Qi WZ, Jiang DX, Chen WZ, Duan N. CodeRetriever: A large scale contrastive pre-training method for code search. In: Proc. of the 2022 Conf. on Empirical Methods in Natural Language Processing. Abu Dhabi: Association for Computational Linguistics, 2022. 2898–2910. [doi: 10.18653/v1/2022.emnlp-main.187]

[84] Guo DY, Ren S, Lu S, Feng ZY, Tang DY, Liu SJ, Zhou L, Duan N, Svyatkovskiy A, Fu SY, Tufano M, Deng SK, Clement C, Drain D, Sundaresan N, Yin J, Jiang DX, Zhou M. GraphCodeBERT: Pre-training code representations with data flow. In: Proc. of the 9th Int'l Conf. on Learning Representations. ICLR, 2021.

[85] Jiang X, Zheng ZR, Lyu C, Li L, Lyu L. TreeBERT: A tree-based pre-trained model for programming language. In: Proc. of the 37th Conf. on Uncertainty in Artificial Intelligence. Toronto: PMLR, 2021. 54–63.

[86] Wang X, Wang YS, Wan Y, Wang JW, Zhou PY, Li L, Wu H, Liu J. CODE-MVP: Learning to represent source code from multiple views with contrastive pre-training. In: Proc. of the 2022 Findings of the Association for Computational Linguistics. Seattle: Association for Computational Linguistics, 2022. 1066–1077. [doi: 10.18653/v1/2022.findings-naacl.80]

[87] Kanade A, Maniatis P, Balakrishnan G, Shi KS. Pre-trained contextual embedding of source code. arXiv:2001.00059, 2020.

[88] Kanade A, Maniatis P, Balakrishnan G, Shi KS. Learning and evaluating contextual embedding of source code. In: Proc. of the 37th Int'l Conf. on Machine Learning. Vienna: PMLR, 2020. 5110–5121.

[89] Chirkova N, Troshin S. CodeBPE: Investigating subtokenization options for large language model pretraining on source code. In: Proc. of the 11th Int'l Conf. on Learning Representations. Kigali: ICLR, 2022. 1–13.

[90] Bui NDQ, Yu YJ, Jiang LX. InferCode: Self-supervised learning of code representations by predicting subtrees. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 1186–1197. [doi: 10.1109/ICSE43902.2021.00109]

[91] Wang X, Wang YS, Mi F, Zhou PY, Wan Y, Liu X, Li L, Wu H, Liu J, Jiang X. SynCoBERT: Syntax-guided multi-modal contrastive pre-training for code representation. arXiv:2108.04556, 2021.

[92] Jia JH, Srikant S, Mitrovska T, Gan C, Chang SY, Liu SJ, O'Reilly UM. CLawSAT: Towards both robust and accurate code models. In: Proc. of the 2023 Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Macao: IEEE, 2023. 212–223. [doi: 10.1109/SANER56733.2023.00029]

[93] Pashakhanloo P, Naik A, Wang YP, Dai HJ, Maniatis P, Naik M. CodeTrek: Flexible modeling of code using an extensible relational representation. In: Proc. of the 10th Int'l Conf. on Learning Representations. ICLR, 2022. 1–23.

[94] Khan J, Uddin G. Automatic code documentation generation using GPT-3. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: Association for Computing Machinery, 2022. 174. [doi: 10.1145/3551349.3559548]

[95] Chen B, Zhang FJ, Nguyen A, Zan DG, Lin ZQ, Lou JG, Chen QZ. CodeT: Code generation with generated tests. In: Proc. of the 11th Int'l Conf. on Learning Representations. Kigali: ICLR, 2022.

[96] Trummer I. CodexDB: Synthesizing code for query processing from natural language instructions using GPT-3 Codex. Proc. of the VLDB Endowment, 2022, 15(11): 2921–2928. [doi: 10.14778/3551793.3551841]

[97] Prenner JA, Robbes R. Automatic program repair with OpenAI's Codex: Evaluating QuixBugs. arXiv:2111.03922, 2021.

[98] Li XN, Guo DY, Gong YY, Lin Y, Shen YL, Qiu XP, Jiang DX, Chen WZ, Duan N. Soft-labeled contrastive pre-training for function-level code representation. In: Proc. of the 2022 Findings of the Association for Computational Linguistics. Abu Dhabi: Association for Computational Linguistics, 2022. 118–129. [doi: 10.18653/v1/2022.findings-emnlp.9]

[99] Bui NDQ, Yu YJ, Jiang LX. Self-supervised contrastive learning for code retrieval and summarization via semantic-preserving transformations. In: Proc. of the 44th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. New York: Association for Computing Machinery, 2021. 511–521. [doi: 10.1145/3404835.3462840]

[100] Niu CA, Li CY, Ng V, Ge JD, Huang LG, Luo B. SPT-Code: Sequence-to-sequence pre-training for learning source code representations. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Pittsburgh: IEEE, 2022. 1–13. [doi: 10.1145/3510003.3510096]

[101] Tipirneni S, Zhu M, Reddy CK. StructCoder: Structure-aware Transformer for code generation. arXiv:2206.05239, 2022.

[102] Ma W, Zhao MJ, Xie XF, Hu Q, Liu SQ, Zhang J, Wang WH, Liu Y. Is self-attention powerful to learn code syntax and semantics? arXiv:2212.10017, 2022.

[103] Liu ZX, Huang Q, Xia X, Shihab E, Lo D, Li SP. SATD detector: A text-mining-based self-admitted technical debt detection tool. In: Proc. of the 40th Int'l Conf. on Software Engineering: Companion Proc. Gothenburg: Association for Computing Machinery, 2018. 9–12. [doi: 10.1145/3183440.3183478]

[104] Wu HJ, Zhang Z, Wang SW, Lei Y, Lin B, Qin YH, Zhang HY, Mao XG. Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques. In: Proc. of the 32nd IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Wuhan: IEEE, 2021. 378–389. [doi: 10.1109/ISSRE52982.2021.00047]

[105] Wang J, Xiao H, Zhong SW, Xiao YH. DeepVulSeeker: A novel vulnerability identification framework via code graph structure and pre-training mechanism. Future Generation Computer Systems, 2023, 148: 15–26. [doi: 10.1016/j.future.2023.05.016]

[106] Liu AQ, Li YZ, Xie XF, Liu Y. CommitBART: A large pre-trained model for github commits. arXiv:2208.08100, 2022.

[107] Yang Z, Shi JK, He JD, Lo D. Natural attack for pre-trained models of code. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: Association for Computing Machinery, 2022. 1482–1493. [doi: 10.1145/3510003.3510146]

[108] Svyatkovskiy A, Fakhoury S, Ghorbani N, Mytkowicz T, Dinella E, Bird C, Jang J, Sundaresan N, Lahiri SK. Program merge conflict resolution via neural Transformers. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. Singapore: Association for Computing Machinery, 2022. 822–833. [doi: 10.1145/3540250.3549163]

[109] Ding YRB, Wang ZJ, Ahmad WU, Ramanathan MK, Nallapati R, Bhatia P, Roth D, Xiang B. CoCoMIC: Code completion by jointly modeling in-file and cross-file context. arXiv:2212.10007, 2022.

[110] Gong Z, Guo YP, Zhou PY, Gao CY, Wang YS, Xu ZL. MultiCoder: Multi-programming-lingual pre-training for low-resource code completion. arXiv:2212.09666, 2022.

[111] Lu S, Duan N, Han H, Guo DY, Hwang SW, Svyatkovskiy A. ReACC: A retrieval-augmented code completion framework. In: Proc. of the 60th Annual Meeting of the Association for Computational Linguistics. Dublin: Association for Computational Linguistics, 2022. 6227–6240. [doi: 10.18653/v1/2022.acl-long.431]

[112] Li J, Li G, Li Z, Jin Z, Hu X, Zhang KC, Fu ZY. CODEEDITOR: Learning to edit source code with pre-trained models. ACM Trans. on Software Engineering and Methodology, 2023, 32(6): 143. [doi: 10.1145/3597207]

[113] Chen ZZ, Yan M, Xia X, Liu ZX, Xu Z, Lei Y. Research progress of code naturalness and its application. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 3015–3034 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6355.htm [doi: 10.13328/j.cnki.jos.006355]

[114] Jiang JJ, Chen JJ, Xiong YF. Survey of automatic program repair techniques. Ruan Jian Xue Bao/Journal of Software, 2021, 32(9): 2665–2690 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6274.htm [doi: 10.13328/j.cnki.jos.006274]

[115] Christopoulou F, Lampouras G, Gritta M, Zhang GC, Guo YP, Li ZQ, Zhang Q, Xiao M, Shen B, Li L, Yu H, Yan L, Zhou PY, Wang X, Ma YC, Iacobacci I, Wang YS, Liang GT, Wei JS, Jiang X, Wang QX, Liu Q. PanGu-Coder: Program synthesis with function-level language modeling. arXiv:2207.11280, 2022.

[116] Shcherban S, Liang P, Li ZY, Yang C. Multiclass classification of four types of UML diagrams from images using deep learning. In: Proc. of the 33rd Int'l Conf. on Software Engineering and Knowledge Engineering. KSIR Virtual Conf. Center: SEKE, 2021. 57–62. [doi: 10.18293/SEKE2021-185]

[117] Zhang W, Luan SM, Tian LQ. A rapid combined model for automatic generating Web UI codes. Wireless Communications and Mobile Computing, 2022, 2022: 4415479. [doi: 10.1155/2022/4415479]

[118] Hossain S, Emi MA, Hossain Mishu M, Zannat R, Ohidujjaman. Code generator based on voice command for multiple programming language. In: Proc. of the 12th Int'l Conf. on Computing Communication and Networking Technologies. Kharagpur: IEEE, 2021. 1–5. [doi: 10.1109/ICCCNT51525.2021.9579880]

[119] Ni C, Xia X, Lo D, Chen X, Gu Q. Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. IEEE Trans. on Software Engineering, 2022, 48(3): 786–802. [doi: 10.1109/TSE.2020.3001739]

[120] Hu B, Wu YJ, Peng X, Sha CF, Wang XC, Fu BQ, Zhao WY. Predicting change propagation between code clone instances by graph-based deep learning. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Program Comprehension (ICPC). Pittsburgh: Association for Computing Machinery, 2022. 425–436. [doi: 10.1145/3524610.3527766]

附中文参考文献:

[113] 陈浙哲, 鄢萌, 夏鑫, 刘忠鑫, 徐洲, 雷晏. 代码自然性及其应用研究进展. 软件学报, 2022, 33(8): 3015–3034. http://www.jos.org.cn/1000-9825/6355.htm [doi: 10.13328/j.cnki.jos.006355]

[114] 姜佳君, 陈俊洁, 熊英飞. 软件缺陷自动修复技术综述. 软件学报, 2021, 32(9): 2665–2690. http://www.jos.org.cn/1000-9825/6274.htm [doi: 10.13328/j.cnki.jos.006274]

宫丽娜(1985－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为智能软件工程, 深度学习, 软件仓库挖掘, 软件分析与测试.

姜淑娟(1966－), 女, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为软件分析与测试, 编译技术.

周易人(1999－), 男, 硕士生, 主要研究领域为软件工程, 深度学习, 软件供应链安全.

魏明强(1985－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为计算机图形学, 3D 视觉, 软件工程, 深度学习.

乔羽(1999－), 男, 硕士生, CCF 学生会员, 主要研究领域为软件工程, 深度学习, 软件缺陷预测.

黄志球(1965－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程, 系统软件, 形式化方法.