无名稿件编号（将由编辑器插入）

为问题添加标签：在问题跟踪系统中应用 API 领域标签

Fabio Santos . Joseph Vargovich . Bianca Trinkenreich . Italo Santos . Jacob Penney . Ricardo Britto . Jo˜ao Felipe Pimentel . Igor Wiese . Igor Steinmacher . Anita Sarma. MarcoA.Gerosa

收稿日期：日期 / 接受日期：日期

摘要 使用完成任务所需的技能为问题添加标签，有助于贡献者在开源软件项目中选择任务。然而，手动为问题添加标签既耗时又容易出错，而当前的自动化方法大多局限于将问题分类为缺陷/非缺陷。我们研究了自动使用我们称为“API 领域”的标签为问题添加标签的可行性和相关性；API 领域是 API 的高层次类别。因此，我们提出，受问题影响的源代码中所使用的API，可以作为处理该问题所需技能类型（例如数据库、安全、用户界面）的代理。我们开展了一项用户研究（n=74），以评估API 领域标签对潜在贡献者的相关性；利用问题描述和项目历史构建预测模型；并通过这些项目的贡献者（n=20）验证预测结果。我们的结果表明：（i）项目新手认为API 领域标签有助于选择任务；（ii）标签预测的平均精确率为84%，召回率为 78.6%；（iii）在一个项目上进行训练、在另一个项目上进行测试（迁移学习）时，预测结果的精确率最高达到 71.3%，召回率最高达到 52.5%；并且

Fabio Santos, Joseph Vargovich, Bianca Trinkenreich, Italo Santos, Jacob Penney, Jo˜ao Felipe Pimentel, Igor Steinmacher, Marco A. Gerosa 北亚利桑那大学 电子邮箱：fabio santos@nau.edu, joseph vargovich@nau.edu, bianca trinkenreich@nau.edu, italo santos@nau.edu, jacob penney@nau.edu, joao.pimentel@nau.edu, igor.steinmacher@nau.edu, marco.gerosa@nau.edu Igor Wiese，巴拉那联邦科技大学 电子邮箱：igor@utfpr.edu.br Anita Sarma 俄勒冈州立大学 电子邮箱：anita.sarma@oregonstate.edu Ricardo Britto 爱立信——布莱金厄理工学院 电子邮箱：ricardo.britto@ericsson.com

arXiv:2304.02877v1 [cs.SE] 6 Apr 2023

(iv) 项目贡献者认为，大多数预测结果有助于识别所需技能。这些发现表明，我们的方法可以应用于实践中，自动为问题添加标签，帮助开发者找到与其技能更匹配的任务。

关键词 API 识别 · 标注 · 打标签 · 技能 · 多标签分类 · 软件仓库挖掘

1 引言

在开源软件（OSS）项目中选择要参与的任务可能颇具挑战性 [1, 2, 3, 4, 5]。未完成的任务会在问题跟踪器中公开报告，但由于问题的复杂度和所需技能各不相同，贡献者可能难以选择合适的任务来承担，尤其是项目新手 [6, 7, 8, 9]。为问题添加标签（也称为“任务”“工单”和“错误报告”）是帮助新贡献者选择工作重点的有效方式 [10]。这种标注策略能够支持各类贡献者，包括新手（新贡献者）、活跃贡献者和维护者，因为他们对该策略重要性的看法相似 [11]。无论此前经验如何，开发者每次开始参与一个新项目时都是新手。然而，社区管理员和项目维护者发现，手动为问题添加标签既困难又耗时 [12]。

我们提出，底层 API（源代码中所需并导入的库）可以经过解析，以指示处理某个问题所需的技能。API 被定义为“一组函数和过程，用于创建能够访问操作系统、应用程序或其他服务的资源或数据的应用程序” [13]。如果贡献者了解代码中使用了哪些类型的 API 来解决问题，他们就可以选择更符合自身技能，或涉及他们希望学习的技能的任务。我们借鉴了这样的观点：API 封装了具有特定用途的模块（例如密码学、数据库访问和日志记录），并将细节从实现中抽象出来。在本研究中，我们聚焦于 API 领域标签：用于指示 API 类别的高层级标签，例如“用户界面”“安全”和“测试”，这些标签可能与处理问题所需的技能相关。

本文扩展了我们此前的工作 [14], ，在该工作中，我们通过对单个项目进行案例研究，探究自动为问题添加 API 领域标签的可行性。在案例研究中首次运行预测后，我们发现，由于缺少关联问题和 PR 的相关信息，数据集中的行数大幅减少。考虑到这一点，为了提升泛化能力，当项目因剔除未关联 PR 的问题而受到数据集大小的严重影响时，我们寻找可行的方法来扩充数据集。此外，在工业项目中，有时会因保密要求而无法访问源代码。基于这一思路，我们探索了在没有可用训练数据时通过迁移学习继续预测 API 领域标签的方法。因此，我们认为

如果 API 领域标签能够应用于许多开源项目或工业项目，无论其源代码数据集的大小或可用性如何，这些标签都会更有帮助。我们通过以下方式扩展了这项工作：（1）将研究范围扩大到五个项目，这些项目使用不同的编程语言、词汇（自然语言）和问题跟踪系统（ITS）；（2）在我们的方法中加入 BERT 技术；（3）扩展定性分析；（4）探索跨项目迁移学习的能力；（5）邀请解决了相关问题的开发者评估 API 领域标签。

我们回答以下研究问题：

研究问题***1***：API 领域标签与新贡献者有多大关联？研究问题***2***：我们在多大程度上能够自动为问题添加 API 领域标签？研究问题***2.1***：我们在多大程度上能够使用项目的数据自动为问题添加 API 领域标签？研究问题***2.2***：我们在多大程度上能够使用其他项目的数据自动为问题添加 API 领域标签？研究问题***2.3***：我们在多大程度上能够使用迁移学习自动为问题添加 API 领域标签？研究问题***3***：API 领域标签与解决问题所需的技能匹配程度如何？

本文研究了这种标注策略与新贡献者的相关性（研究问题1）。我们创建了模型并评估其性能。通常，机器学习方法使用同一项目的数据进行训练和测试（研究问题2.1）。然而，当现有数据不足以创建达到预期性能的预测模型时，可以考虑扩大数据集，将其他项目纳入其中（研究问题2.2）。此外，如果目标项目完全缺少历史训练数据，可以使用同一领域中经过预训练的数据集（源项目）在目标项目中运行预测 [15]（研究问题2.3）。因此，我们还开展了迁移学习研究。最后，我们研究了开发者对预测结果的看法，以确定 API 领域标签是否能够充分指示技能，并帮助新手选择任务（研究问题3）。

我们的贡献包括：(1) 新手如何看待 API 领域标签的相关性；(2) 一种新的半自动 API 分类流程；(3) 一种预测以多种编程语言（C、C# 和 Java）编写、且问题使用葡萄牙语和英语的项目所需技能的机制；以及 (4) 与开发者共同验证 API 领域标签。

2 相关工作

整理问题需要付出一些标注工作量。标注对于描述功能，以及更轻松、更快速地理解和搜索软件工件 [11]非常重要。不过，手动为软件工件标注

可能会很困难且耗时。因此，有人提出了一些方法，用于自动标注软件项目 [16] 和依赖项 [17]。虽然这些方法证明了对软件工件进行标注的可行性，但它们是在更高层次上对整个项目进行分类。相比之下，我们的方法对较小的软件工件（即问题和 API）进行分类。

也有人提出了为其他软件工件添加标签的方法，例如为 Stack Overflow 上的问题添加标签 [18, 19, 20]。Xia 等人 [18] 根据问题与以往问题的相似度，为问题推荐标签。这些方法仅限于使用现有标签，无法用于问题跟踪系统或 API。Uddin 和 Khomh [20] 以及 Lin 等人 [19] 为用户对 API 的评价添加标签。尽管他们的研究聚焦于 API，但目标是支持开发者决定是否采用新的 API。在这项工作中，我们的目标恰恰相反。鉴于项目已经涉及不同领域的 API，我们旨在帮助开发者找到包含他们更熟悉的 API 的任务。

尽管许多方法旨在为问题添加标签，但大多数方法只尝试区分错误报告与非错误报告 [21, 22, 23, 24, 25, 26]。能够依据其他标签进行分类的方法寥寥无几 [27, 28, 29]。其中，Izadi 等人 [28]和 Wang 等人 [29] 使用文本分类算法 BERT[30] 处理多个标签，我们也采用了这种方法。尽管这些方法能够将内容分为不同标签，但它们仅使用预先存在的标签进行分类。我们的方法不使用从现有问题中提取的预定义标签，也不使用 GitHub 默认提供的标签，而是根据 API 领域定义标签。这种标注有助于引导新贡献者了解可以贡献什么[31, 10], ；若没有指引，这可能会是一项艰巨的任务 [2]。

为实现类似的目标、支持新贡献者，GitHub 等社交编程平台[1] 鼓励项目为[2] 适合新贡献者的问题添加标签，这一做法已被多个社区采用（例如 LibreOffice、[3] KDE、[4], 和 Mozilla[5]）。然而，社区管理员认为，手动为问题添加标签既困难又耗时 [12]。因此，Huang 等人 [32] 提出了一种为适合新手的优先问题添加标签的方法。虽然这种方法能为新贡献者标出简单的问题，但其结果的局限性与仅将问题分类为缺陷的方法一样。相比之下，通过为问题标注 API 领域，我们的方法能够支持不同技能水平的新贡献者。

3 方法概述

本节概述我们如何回答这些研究问题。

RQ.1: API领域标签对新贡献者的相关性如何？ 在此研究问题（第 4 节）中，我们评估了经过人工整理的标签及其潜在

[1] http://bit.ly/NewToOSS[2] 在本研究中，“任务”和“问题”这两个词交替使用。[3] https://wiki.documentfoundation.org/Development/EasyHacks

[4] https://community.kde.org/KDE/Junior_Jobs

[5] https://wiki.mozilla.org/Good_first_bug

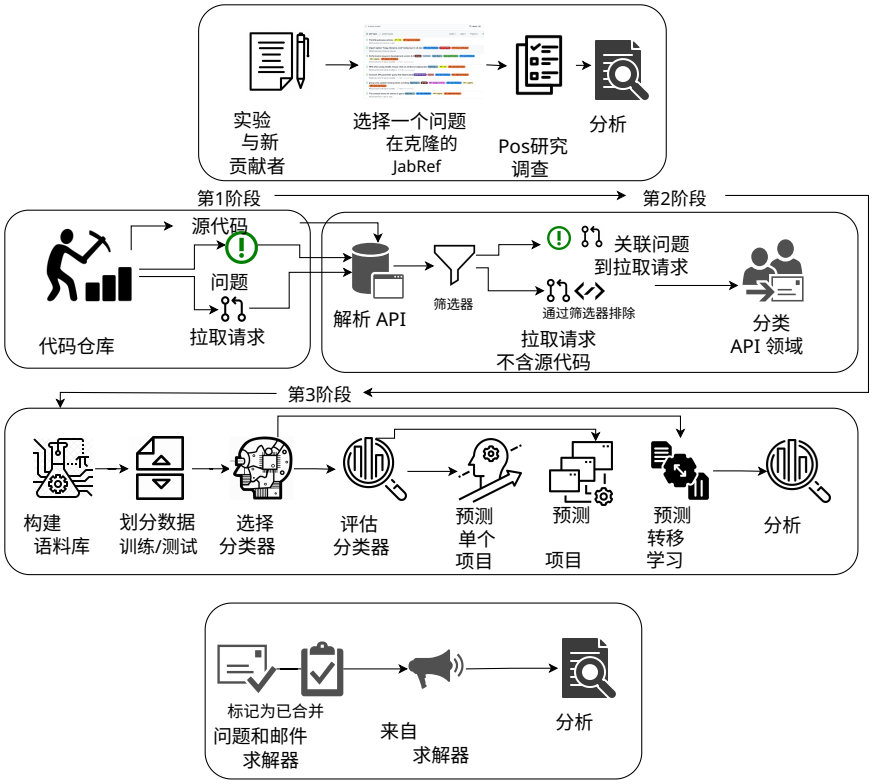


图1 研究方法概述

新贡献者。我们将参与者分为两组。在模拟该项目的 22 个问题页面后，我们为实验组的问题添加了 API领域标签，并让对照组的页面保持原样。我们请参与者选择要贡献的问题，并填写一份关于其选择过程的调查（图1 - RQ1）。

研究问题***2***：我们能在多大程度上自动为问题标注 *API* 领域标签？在此研究问题中（第 5 节），我们探讨预测 API 领域标签的可行性。我们挖掘了软件仓库，以收集问题、与其相关联的拉取请求，以及源代码中使用的 API。随后，我们手动将 API 分类到各个 API 领域，以构建机器学习分类器（图1 - RQ2）。为回答子问题，我们分别使用每个项目的数据集（研究问题2.1）、合并所有项目的数据集（研究问题2.2），以及不同的源数据集和目标数据集（研究问题2.3）来预测 API 领域标签。

RQ.3：*API* 领域标签与解决问题所需的技能匹配度如何？最后，在这个研究问题（第 6 节）中，我们要求贡献者提供

关于我们预测的标签在识别完成问题所需技能方面实用性的反馈（图1 - RQ3）。

为促进可复现性，我们提供公开可用的补充材料[6]，其中包含原始数据、Jupyter 笔记本脚本以及匿名化的调查数据。

4 标签对新贡献者的相关性（RQ1）

4.1 方法

为了从外部人士的视角探究 API领域标签的相关性，我们开展了一项有74名参与者的实验。我们选择托管在 GitHub上的 JabRef 项目作为实验对象。本文的两位作者已经参与过该项目，并且对项目有深入了解。这些知识帮助我们解读反馈和结果。我们创建了两个版本的 JabRef 问题页面（有API 领域标签和无 API 领域标签），并将参与者分为两组（被试间设计）。我们要求参与者选择并排列他们愿意贡献的三个问题，并回答一份后续问卷，说明哪些信息支持了他们的决定。本阶段使用的工件属于复现包的一部分。

4.1.1 参与者

我们采用便利抽样，从业界和学术界招募参与者。我们联系了个人和职业网络中的讲师和信息技术经理，请他们协助邀请参与者。在业界方面，我们从一家设在巴西的中型信息技术初创企业以及一家大型全球性公司的信息技术部门招募了参与者。我们从多所大学招募了学生，包括美国一所大学和巴西另外两所大学的计算机科学本科生和研究生，以及巴西一所大学的数据科学研究生，因为他们也是JabRef项目的潜在贡献者。表 1介绍了参与者的人口统计学信息。我们提供了一张亚马逊礼品卡（美国$ 25.00）作为参与激励。

我们将参与者的开发任职年限分为新手和经验丰富的程序员，并以平均“专业开发者从业年限”为界，将样本分为两组——低于和高于平均值。我们还将参与者划分为行业从业者和学生。参与者以顺序编号标识（“参与者”列）。

参与者被随机分为两组：对照组和处理组。在开始填写问卷的 120 名参与者中，有 74 名（61.7%）完成了所有步骤；我们仅在分析中考虑这些参与者。 We

[6] https://doi.org/10.5281/zenodo.6869246

表 1 实验参与者的人口统计学子组

总体	数量	百分比
业界	41	55.5 %
学生	33	44.5 %

任职年限	数量	百分比
专家	19	25.7 %
新手	55	74.3 %

最终对照组和实验组分别有 33 名和 41 名参与者。

4.1.2 实验规划

我们选取了 22 个现有的 JabRef 问题，并为对照组和处理组创建了模拟 GitHub 页面。我们从最新的问题中进行选取，尽量使每个问题预测出的 API 领域标签数量分布以及预测出的 API 领域标签计数保持相似。对照组的模拟页面仅包含 JabRef 问题原有的标签，而处理组的模拟页面除了原有标签外，还展示了 API 领域标签。这些页面可在复现包中获取。我们使用预测模型的初步版本生成了 API 领域标签 [14]。

4.1.3 问卷数据收集

问卷包含以下问题/说明：

– 选择你想处理的三个问题。– 选择问题页面中帮助你决定选择哪些问题的信息（区域）（图：2）。– 你选择的信息为何相关？（开放式问题）– 选择你认为有助于挑选这三个问题的标签。– 你希望在问题中看到哪种标签？（开放式问题）

问卷还询问了参与者的经验水平、作为开源软件贡献者的经验，以及在 JabRef 所用技术方面的专业水平。

图 2 展示了问题详情页面和问题列表页面上的问题条目示例。选择要贡献的问题后，参与者会看到此页面，以选择与其所选问题相关的信息区域。

4.1.4 问卷数据分析

我们将分析分为两组问题。

区域和标签选择分析。我们首先比较了实验组和对照组的结果，以了解参与者对哪些内容的感知，关于 what



图 2 关于页面区域与任务选择相关性的问卷问题

他们认为重要的信息区域以及 API-领域标签的相关性。我们使用小提琴图直观地比较分布，并使用 Cliff's Delta 检验测量效应量。

随后，我们分析了数据，根据参与者的人口统计信息对其进行汇总，得出了表 1 中所呈现的子组。我们计算了优势比，以检查从两个组获得相似回答的可能性有多大。对于每项比较，我们都使用了2×2列联表——例如，行业从业者与学生，以及有经验的程序员与新手程序员。我们使用以下公式计算优势比 [33]:

$$OddsRatio(OR) = \frac{(a/c)}{(b/d)}$$

优势比 > 1 意味着第一个子组更有可能报告某种类型的标签，而优势比小于 1 意味着第二个组的机会更大（ OR ） [34]。

开放式问题分析。为了理解标签选择背后的依据，我们对开放式问题的答案进行了定性分析（“为什么你选择的信息具有相关性？”以及“你希望在问题中看到什么类型的标签？”）。我们选取了具有代表性的引述，以说明参与者对标签相关性的感知。

我们通过在组中归纳应用开放式编码，对回答进行了定性分析。在此过程中，我们识别了参与者认为所提供信息相关的原因，以及参与者希望获得哪些信息。随着分析的推进，我们逐步构建了后续形成的代码，并将其与转录文本的相应部分关联起来，从参与者的视角对信息相关性进行编码。

研究人员每周会面，讨论编码。我们讨论了代码及其分类，直到就代码的含义以及代码之间的关系达成共识。最终形成了一组高层级类别，如我们的编码手册中所编录的那样[7]。

4.2 结果

选择任务时使用的信息。了解参与者在选择问题时作出决策所使用的信息类型，有助于项目更好地在问题页面上组织此类信息。图 3 展示了参与者认为有用的不同区域。在对照组中，最受关注的两个区域包括问题的标题（78.8 %）和正文（75.8%），其次是标签（54.5%）。这表明项目使用的标签仅具有边际效用，参与者不得不查看问题详情。相比之下，在实验组中，按优先级排列的前三个关注区域分别为：标题、标签和正文（分别为 97.6%、82.9 % 和 70.7%）。这表明实验组的参与者认为标签比对照组的参与者更有用：实验组中的使用率为 82.9%，而对照组为 54.5%。比较两组中的正文和标签区域后，我们发现实验组参与者选择标签区域的次数是对照组的 1.6 倍（$p<0.05$）。

我们的定性分析表明，标签有助于筛选问题。例如，P2 提到：“标签有助于了解问题领域；读完问题标题后，标签是首先考虑的内容，甚至早于打开问题查看详情”。参与者认为，标签有助于识别问题的具体主题，正如 P4 所说：“[标签是]提示，说明哪些领域与当前出现的问题有关”。

[7] https://doi.org/10.5281/zenodo.6869246

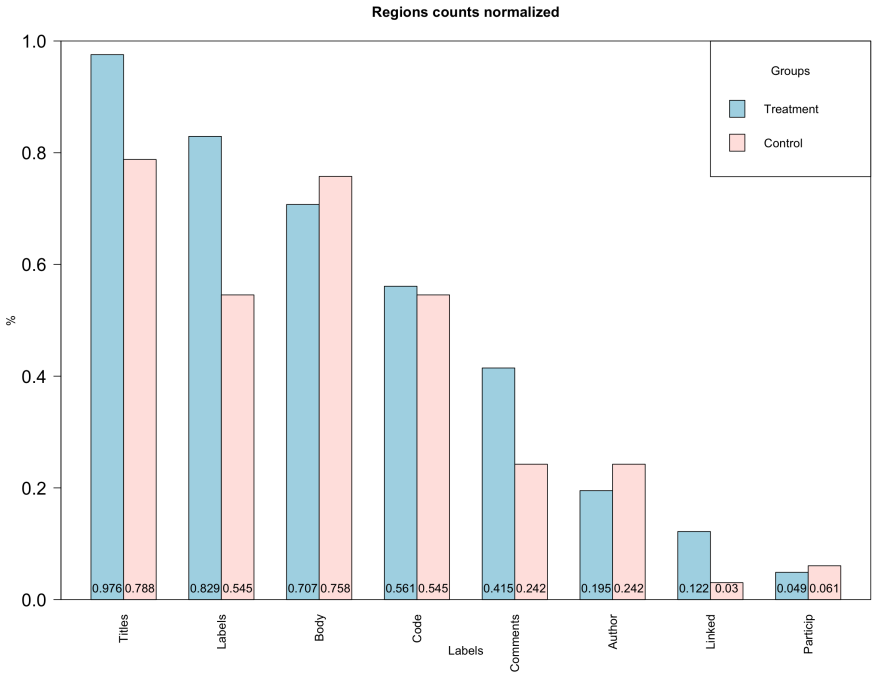


图3 参与者从实验组和对照组中选出的最相关问题信息页面的区域计数（已标准化）。

标签的作用。我们还调查了哪些类型的标签有助于参与者做出决策。我们根据这些标签所包含的信息类型，将参与者可用的标签分为三组。

–问题类型（项目中已存在）：包含有关任务类型的信息，如缺陷、增强、功能、适合新手的问题和 GSoC（Google Summer of Code）。– 代码组件（项目中已存在）：包含有关 JabRef 特定代码组件的信息：条目、组、外部文件、主表、抓取器、条目编辑器、首选项、导入、关键词– API领域（新标签）：由我们的分类器生成的标签（输入输出、用户界面、网络、安全等）。这些标签仅对处理组开放。

表2 对照组和处理组中的标签分布

标签类型	对照组	**百分比**	**实验组**	**百分比**
问题类型	145	56.4 %	168	36.8 %
组件	112	43.6 %	94	20.6 %
API领域	-	-	195	42.7 %

表3 不同人口统计学子群体关于 API 标签（API/- 组件/问题类型）的回答

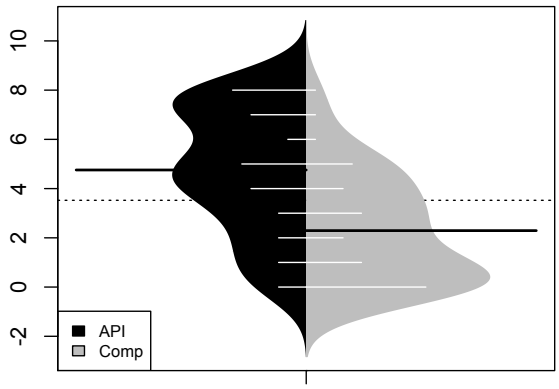
子组	比较	API %	组件 或 类型 %
业界	API/组件	**56.0**	44.0
学生	API/组件	40.0	**60.0**
有经验的程序员	API/组件	**50.9**	49.1
新手程序员	API/组件	41.5	**58.5**
业界	API/问题 类型	45.5	**55.5**
学生	API/问题 类型	30.6	**69.4**
有经验的程序员	API/问题类型	43.5	**56.5**
新手程序员	API/问题 类型	30.9	**69.1**

表2比较了参与者认为相关的标签（第 4.1.3 节），这些标签分属不同类型，并在实验组和对照组之间进行分布。在对照组中，参与者选择最多的标签（56.4%）与问题类型有关（例如，缺陷或增强）。然而，在实验组中，这一比例降至 36.8%，API 领域标签占多数（42.7%），其次是代码组件标签（20.6%）。这种分布差异表明 API 领域标签具有实用价值。

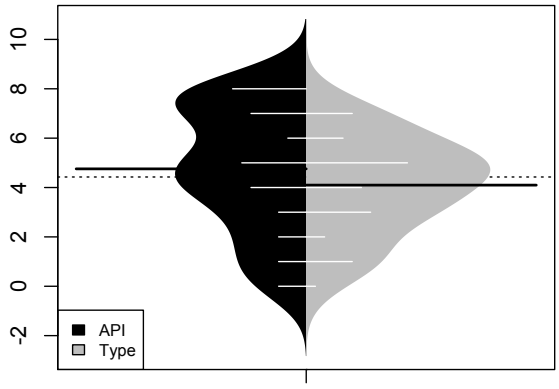
为了更好地理解 API领域标签相较于其他类型标签的实用性，我们进一步调查了处理组参与者对标签的选择。图4展示了两个小提琴图，比较了（a）API领域标签与代码组件标签，以及（b）API领域标签与问题类型。小提琴图中较宽的部分表示观测值取某一给定值的概率较高，较窄的部分则对应较低的概率。图中显示，与代码组件标签（中位数为2个标签）相比，API领域标签被更频繁地选择（中位数为5个标签），且具有较大的效应量（$|\mathrm{d}| = 0.52$）。然而，问题类型和API 领域标签的分布相似，这一点也得到可忽略效应量（$|\mathrm{d}| = 0.1$）的证实。这些结果表明，尽管问题类型（缺陷修复、增强、适合新手）很重要，但理解解决任务所需的技术（API）要求，对于开发者决定选择哪个任务同样重要。

最后，我们分析了人口统计学子群体对 API 领域标签是否存在不同看法（表3）。比较业界与学生时，我们发现，在控制组件标签后，业界参与者选择 API 领域标签的数量是学生的 1.9 倍（p 值=0.001）。在控制问题类型后，我们发现优势比相同（p 值=0.0007）。比较有经验的程序员与新手程序员时，在控制组件标签后，我们未发现统计显著性（p=0.11）。然而，在控制问题类型后，我们发现有经验的程序员选择 API 领域标签的数量比新手程序员多 1.7 倍（p 值=0.01）。

优势比分析表明，与学生和新手程序员相比，从业者和有经验的开发者更有可能认为 API 领域标签与其相关。



API × 组件



API × 类型

图 4 Y 轴包含 API 领域标签（API）× 组件标签（Comp）× 类型标签的概率密度和中位数

贡献者分析问题的方式。我们使用问卷中的开放式问题来评估受试者如何利用这些信息来判断该任务是否适合自己（第4节.1.4）。

我们的定性分析发现，贡献者在决定参与哪项任务时，会认为一组共 22 个信息类别与之相关。我们依据文献中的一个现有模型——5W2H 框架，对这 22 个信息类别进行了整理，具体如下文所述，并在图 5 中展示。5W2H 框架（5个W问题和2个How问题）常用于澄清问题、议题、错误或不符合项，或促进有效措施的实施。该框架最初应用于

汽车及其他制造业 [35]，后来又应用于质量管理 [36] 和软件工程 [37]。

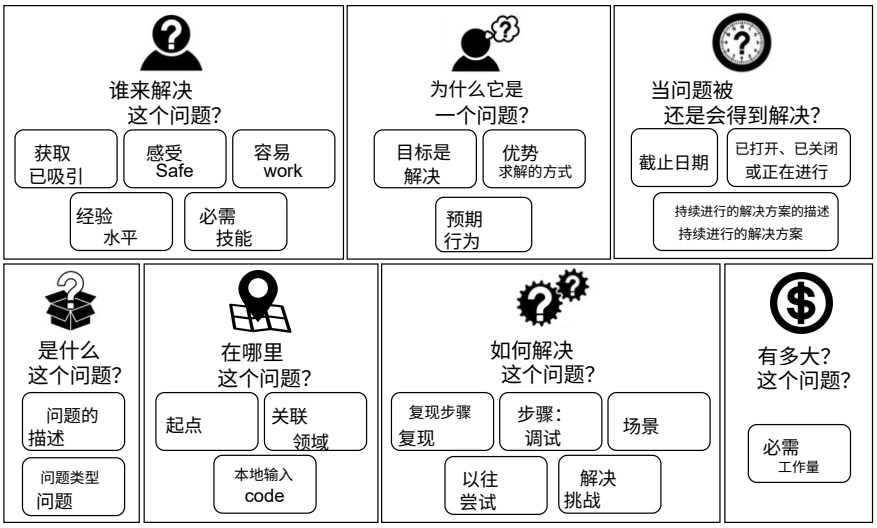


图 5 贡献者报告的与选择任务相关的信息。我们将参与者定义的类别（圆角方框）映射到 5W2H 框架 [37],，该框架通过七个问题组织用于决策的信息。

___ 谁会解决问题？ 此类别包含有关影响人们选择处理问题的因素的信息。贡献者提到了哪些因素会影响一个人选择问题的决定。新手可以在“筛选标签，以搜索他们 *[最想]*贡献的问题”（P34）以及阅读标题（P18）时选择该问题，看看它是否“包含一些不太冗长的内容，并且使用了 *[他们]* 能够轻松理解的词语”（P21）。打开问题时，参与者还表示，正文和评论与“引起对该问题的兴趣”（P4）有关，因为“项目中经验丰富的人提供的详细正文和有帮助的评论，对于让新手 FEELING SAFE 尝试处理该问题非常有帮助”（P6）。当贡献者将自己的 EXPERIENCE LEVEL 与解决问题的难度指示相匹配时，他们决定是否处理某个问题的信心会增强（P8、P4）；这种指示可以显示在“简单、中等、困难”（P4）、“适合新手的问题”（P6）或“有挑战性的好问题”（P7）等标签中。除了自身经验外，贡献者还可以使用 REQUIRED SKILLS 来处理问题，从而判断“他们是否具备 *[必需的]* 技能来提供帮助”（P31），或者“是否 *[有能力]* 找到解决方案”（P32）。参与者提到的必需技术技能包括代码所使用的编程语言（P21、P27、P33）、架构层——前端、后端、接口（P27、P3、P2）、API（P41、P42）、数据库（P33）、框架和库（P20）。

___ 为什么这是一个问题？ 是一个将问题视为问题的类别。参与者提到，说明问题存在的理由可能会引起新贡献者的兴趣，例如了解 GOAL TO SOLVE 以及“问题的目的是什么”(P44)，以及 BENEFITS OF SOLVING 或“解决它为什么会帮助用户”(P45)。此外，软件的 EXPECTED BEHAVIOR 有助于阐明为什么所报告的问题相较于软件的正常行为而言是一个问题。因此，预期行为代表着“决定系统中正在发生什么以及预期发生什么的关键信息”(P61)。确实，一位参与者表示：“我只会为自己知道其工作原理的东西做贡献”(P22)。

___ 问题何时已解决，或者何时会解决？引出了与问题相关的时间信息和限制条件。参与者表示，他们想知道问题的 DEADLINE TO SOLVE ，以及“紧迫性”（P13）。参与者建议将优先级显示为标签（P17），并根据问题对企业或用户造成的影响来定义（P15）。另一个与时间相关的问题是“用于检查问题状态的状态”（P33），可以是 OPEN、 CLOSED、 OR ONGOING，以便贡献者在问题页面使用筛选器。由于他们“不会经常查看已关闭的问题， [...]开放标记会吸引 [他们] 的注意”（P43）。当贡献者正在处理解决方案时，应将其姓名分配给该问题，并添加一条评论，其中包含 DESCRIPTION OF AN ONGOING SOLUTION ，以“展示问题的状态”（P35）。

___ 什么是问题？ 涉及问题本身的描述。参与者提出了清晰的 ISSUES' DESCRIPTION的重要性，包括概括性的“问题所涉及内容的概念”(P28)以及全面的解释“以帮助理解问题所在”(P45)。当一个问题同时提供这两个层次的细节时，它“介绍问题，先是用一般术语，随后给出 *[它们]* 的详细信息”(P12)。问题的 TYPE 在标签中“展示 *[...]* 问题 *[是如何]* 分类的”(P35)。参与者建议问题应具有“能够准确说明问题属于哪种类型的标签”(P40)：缺陷(P41)、新功能(P42)、性能(P42)、增强(P42)和安全。一位参与者(P43)强调“‘所有问题都应具有类型，这样 *[他们]*就能看到 *[他们的]* 技能组合是否有用”(P43)。

___问题在哪里？指的是问题在代码或项目中的定位，可引导贡献者找到 START POINT 或“在代码/库中的哪里着手调查问题”（P4）。 LOCAL IN CODE 或者导致问题的代码块、方法或类，以及 CONNECTEDAREAS。这些信息会“提供一些线索，指出哪些区域与正在发生的问题有关联”（P4），并提供“代码片段作为上下文，说明程序中的哪个位置发生了此问题”（P18）。

如何解决这个问题？提供切实可行的指导，帮助解决问题。了解“ PREVIOUS ATTEMPTS 解决 *[问题]*”（P30）有助于贡献者“了解已经完成并妥善记录的有价值信息”（P42）。正在考虑是否处理某个问题的贡献者可以阅读“[SOLVING] CHALLENGES”（P35），以避免在先前工作上浪费时间。

尝试并将工作量集中在新的路径上，以实现解决方案。在处理问题时，在受控环境中遇到 STEPS TO REPRODUCE the error (P45) 也有助于解决问题。参与者还提到，他们希望看到“关联问题和评论，以帮助理解SCENARIO”(P33)，并且 STEPS TO DEBUG 能够“弄清问题究竟是什么”(P41)。

问题有多大？是能够让人了解REQUIRED EFFORT 对于“*[一位贡献者]* 独自处理并解决问题需要多久”(P7)。如果问题没有提供这些信息，开发者会尝试“理解问题的想法，以便更好地估算解决问题需要多长时间”(P19)。

最后，“在哪个区域可以找到这些信息？”这个问题确定了本研究参与者找到信息的区域。标题出现了 7 次，正文 8 次，评论 13 次，标签 5 次，状态 2 次，代码片段 3 次，关联问题 2 次。

5W2H 分析结果：分析证实了标题、正文、评论和标签的重要性，并帮助建立了一个分类体系，归纳贡献者在决定是否要参与某个问题时会分析哪些内容。针对这个开放式问题构建的定性编码，可在未来的工作中进一步探索，以便通过模板、标签、机器人或其他用户界面对象向贡献者展示此类信息。

首选的标签类型。为了评估贡献者希望在问题页面中看到的标签，42名参与者（共74人）回答了开放式问题Q2（“你希望在问题中看到什么类型的标签？”）。TYPE、 PRIORITY 要解决的，以及 PROGRAMMING LANGUAGE “编写代码所使用的语言”（P21）是被提及最多的三类，随后是 DIFFICULTY LEVEL、 TECHNOLOGY和 API。一些参与者为标签 TYPE提出了不同的语义：缺陷（P3、P41）、改进（P3）、性能（P35、P42）、新功能（P42）或安全（P36）。其他参与者也为 DIFFICULTY LEVEL提出了不同的语义：“适合新手的问题”（P6）、“具有良好挑战性的问题”（P7）或“简单、中等、困难”（P4）。每个标签的语义可以在未来的工作中进一步探讨。我们在表4中展示了根据参与者的回答进行定性编码后得到的11个建议标签类别。

参与者更倾向于查看有关优先级、类型、编程语言、复杂度、技术和 API 的标签，而不是架构、状态、GitHub 信息、数据库和框架。GitHub 信息是关于项目代码库的一般信息（例如“关于评论最多项目的排名”（P10P0）和“分支”（P22I0））。

表4 参与者希望用于选择问题的标签

组	期望的标签	提及的参与者
管理	Type	P41、P35、P42、P3、P36、P37、P38、P39、P43、P40
管理	优先级	P12, P13, P19, P20, P14, P15, P16, P29, P17, P18
技术	编程 语言	P34, P33, P21, P22, P27, P23, P24, P22, P26
管理	难度 级别	P4、P5、P6、P7、P9、P8
技术	技术	P30, P34, P33, P32, P20, P31
技术	API	P41, P42, P3, P1, P20
技术	架构 层	P2, P3, P27, P18
管理	状态	P28、P9、P29
管理	GitHub 信息	P10、P19
技术	数据库	P33
技术	框架	P20

RQ.1 总结。我们的研究结果表明，标签与选择要处理的问题相关。API 领域标签提升了人们对标签相关性的认知。对于业内人士和有经验的程序员而言，API 领域标签尤为相关。API 是用户希望看到的问题标签之一。5W2H 分析证实了标签的相关性，并可指导贡献者如何撰写问题。

5 标签预测结果（RQ2）

尽管 API 领域标签具有相关性，我们仍研究了如何自动预测这些标签。

5.1 方法

为了预测 API 领域标签，我们采用了一种多标签分类方法。该方法分为三个阶段：第1阶段——挖掘代码仓库；第2阶段——解析源代码，并在专家协助下半自动地对 API 进行分类；第3阶段——构建语料库并运行分类器（图1）。此外，我们还探讨了问题元素（即标题、正文和评论）以及机器学习设置（即 n 元语法和不同算法）对预测结果的影响。

在我们的初步工作中 [14], ，我们针对单个项目（JabRef）开展了一项探索性实验。在当前研究中，我们纳入了四个新项目。我们选择这些项目，以增加领域、编程语言和人类语言（词汇）的多样性。我们希望涵盖一批受欢迎的开源软件（OSS）项目和闭源项目，这些项目目前仍处于活跃状态，并且拥有大量问题和拉取请求。由于我们的目标是在项目社区内开展调查，

表 5 项目详情。R - 发布版本数量。C - 贡献者数量。

项目	R	C	星标	分支	已关闭 拉取请求	问题	领域	OSS
JabRef	42	337	15.7K	1.8K	2.7K	4.1K	文章	Y
Audacity	25	154	6.9K	17.1K	1.1K	0.6K	经理 音频 编辑器	Y
PowerToys	50	262	6.55万	3.7K	3.3K	9.9K	实用工具 适用于 Windows	Y
RTTS	121	40	N.A.	N.A.	N.A.	N.A.	电信- 通信 产品	N
Cronos	123	–	N.A.	N.A.	N.A.	N.A.	Time Tracker	N

我们提前联系了候选项目的维护者/经理，以解释我们的目标，并寻求他们的支持，以联系用户研究的贡献者。表 5 展示了所选项目及其特征。

该研究可分为两个预测分支：TF-IDF 和 BERT。TF-IDF 预测沿用了先前的研究 [14], ，采用五种算法（随机森林、决策树、逻辑回归、MLPClassifier 和多标签 k 近邻），但扩展到了更多项目、集成测试系统、编程语言和词汇（自然语言）。BERT 预测也进行了相同的扩展，但仅限于英语词汇。与 TF-IDF 不同，BERT 根据词语在句子中的上下文来确定语料库中词语的含义。我们在问题标注问题的背景下，将 BERT 与先前的 TF-IDF 分类流程进行了比较。

5.1.1 第1阶段 - 软件仓库挖掘

我们首先从代码仓库中收集数据，用于训练机器学习模型以预测 API 标签。为实现这一目标，我们挖掘了已关闭的问题和已合并的拉取请求。表 6 总结了各项目的特征和人口统计学信息。截至 2021年11月，我们收集了所有项目共计 22,231 个问题和 4,674 个拉取请求（PR）的数据。对于开源软件项目，我们使用 GitHub REST API v3 收集标题、正文、评论和关闭日期等数据。我们还收集了拉取请求中发生更改的文件名称，以及与每次提交相关联的提交信息。工业项目使用 Gerrit（RTTS - 实时电信软件）和 Jira+MTT - Minds At Work Time Tracker（Cronos）。对于 RTTS，我们提取了两个 CSV 文件：其中一个包含“问题”（RTTS 中的故障），另一个包含提交记录。Cronos 项目结合使用 Jira 跟踪未解决的问题，并使用软件 MTT（一种内部解决方案）管理修订和时间分配。我们从 Jira 中提取了一个 CSV 文件，并从 MTT 中提取了一个 TXT 文件。

接下来，为了训练模型，我们只保留了与已合并和已关闭的拉取请求相关联的问题中的数据，因为我们需要将问题数据映射到源

表 6 挖掘的项目和问题跟踪系统

项目	Prog Lang	问题 Tracker/ 词汇- ulary	提取 方法	问题/ PR	已关联 问题 & PR	来源 Code 文件	不同的 APIs
JabRef	Java	GitHub EN	GitHub API V3	4,471 1,966	1,914	1,690	1,944
Audacity	C++	GitHub EN	GitHub API V3	1,440 310	341	624	1,478
PowerToys	C#	GitHub EN	GitHub API V3	12,571 853	1,011	794	264
RTTS	Java	Gerrit EN	导出 CSV	2,836 470	470	9,779	8,645
Cronos	Java	Jira/MTT BR	导出 CSV/TXT	913 1075	206	220	441
总计				22,231 4,674	3,942	13,107	12,772

代码 API。为了在开源项目中查找拉取请求与问题之间的链接，我们在拉取请求的标题和正文中搜索符号 #issue_number，并检查与每个链接关联的 URL。此外，我们还筛除了关联的拉取请求中没有至少一个源代码文件的问题（例如，只关联了文档文件的问题），因为这些问题无法向模型提供任何与 API 相关的内容。同样，对于托管在 Gerrit 和 Jira/MTT 上的项目，我们使用故障 ID 和键字段（Gerrit）进行关联；对于由 Jira/MTT 管理的项目，则使用变更 ID 和修订字段进行关联。我们需要解析 MTT 的 TXT 文件以查找修订字段。我们舍弃了没有源代码或关联数据的条目。总计舍弃了 734 条记录。

5.1.2 第*2*阶段 - *API* 分类

第2阶段涵盖 API 提取和专家分类。

API 提取。 为了识别每个拉取请求所影响的源代码中使用的 API，我们构建了一个解析器，用于处理这些项目中的所有源文件。我们将 13,107 个源文件中的 12,772 条库声明语句映射到 185,159 种文件与 API 之间的可能关系。解析器会查找特定命令，即 import（Java）、using（C#）和 include（C++）。解析器会识别所有类，包括每条 import/using/include 语句中的完整命名空间。我们仅考虑每个项目中使用频率最高的语言。

随后，我们筛除了源代码最新版本中不存在的 API（JabRef 5.3、Audacity 3.1.0 和 PowerToys 0.49.1；RTTS 和 Cronos 是业界项目，我们使用了最后提供的版本），以避免推荐项目中已不再使用的源代码 API。筛选过程是自动的。处理已关闭的拉取请求时，解析器会将其附带文件的文件名与数据库中存储的文件名进行比较。如果数据库中没有找到该文件名，则会将该拉取请求从训练集中剔除。

最终数据集包含 22,231个问题、4,674 个拉取请求、13,107 个文件以及 12,772 个不同的 API（表 6）。

表 7 标签定义

标签已生成	定义
应用程序（App）	附加到特定用途的第三方应用程序或插件系统
应用性能管理器（APM）	监控性能或基准测试
大数据	用于存储大量数据的 API，多种格式
云	用于在互联网上运行的软件和服务的 API
计算机图形学（CG）	操纵视觉内容
数据结构	数据结构模式（例如集合、列表、树）
数据库（DB）	数据库或元数据
软件开发和IT 运维（DevOps）	库、版本控制、持续集成和持续交付
错误处理	错误状态下的响应与恢复流程
事件处理	对事件作出响应的组件，例如监听器
地理信息系统（地理信息系统）	地理定位信息
输入输出（IO）	读取、写入数据
解释器	编译器或解释器功能
国际化（国际化）	整合并融入国际、跨文化及全局维度
逻辑	框架，类似命令的模式，控件或面向架构的类
语言（Lang）	内部语言特性与转换
日志记录	应用程序的日志注册表
机器学习（ML）	ML 支持，例如基于训练数据构建模型
微服务/服务	可独立部署的较小型服务。接口位于两个不同应用程序之间，使它们能够相互通信
多媒体	通过文本、音频、视频表示信息
多线程（线程）	支持并发执行
自然语言处理（自然语言处理）	处理和分析自然语言数据。
网络	Web 协议、套接字、RMI API
操作系统（OS）	用于访问和管理计算机资源的 API
解析器	将数据分解为可识别的部分，以便进一步分析。
搜索	用于网页搜索的 API
安全	密码学和安全协议
设置	内部应用配置
用户界面（UI）	定义表单、屏幕和视觉控件
实用工具（Util）	通用第三方库
Test	测试自动化

专家分类。 三位软件工程专家（资深开发人员），其中包括本文的一位作者，根据他们在软件开发方面的经验提出了这些标签，并考虑了足够通用、适用于软件项目中各种 API 的类别。对于

例如，提议的领域包括用户界面、输入输出、云、错误处理等。经过四轮讨论，专家们达成共识，共定义了 31 个 API 领域（表 7）。

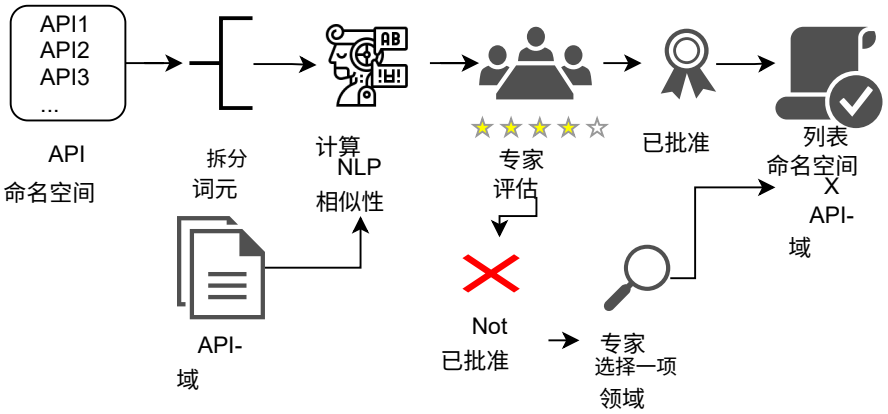


图 6 专家评估 API 的流程

在定义了31 个 API领域后，我们开始对API进行半自动分类（图 6）。API分类方法背后的直觉是，库的命名空间通常会揭示架构信息，并因此揭示其类别或API领域 [38, 39]。为了确定每个API 可能所属的 API领域，我们将所有 API 命名空间拆分为词元。例如，API“com.oracle.xml.util.XMLUtil”被拆分为 “com”、“oracle”、“xml”、“util” 和 “XMLUtil”。接下来，我们去除了业务领域名称扩展（例如，“org”、“com”）、国家代码顶级域名（“au”、“uk”等）以及项目和公司名称（“microsoft”、“google”、“facebook”等）。在该示例中，我们保留了第一个词元 “xml”、第二个词元 “util” 以及完整命名空间 “com.oracle.xml.util.XMLUtil”。

对于每个词元，我们使用自然语言处理相似度函数，确定它与 31 个拟议 API 领域的相似程度。此举旨在为专家推荐可能适用于这些 API 的领域。我们使用了自然语言处理 Python 软件包 spacy[40]。Spacy 是一款多用途自然语言处理软件包，可使用 word2vec 获取词语的语义相似度。我们为 spacy 软件包配置了可用的最大训练模型（完整大型向量软件包 en_core_web_lg，其中包含 685k 个唯一向量）。

为协助专家评估并缩小搜索范围，我们汇总了命名空间中找到的词元。例如，为了评估 Cronos 项目的 API，专家收到了一份包含 32 个“第一词元”的列表，以及一份包含 73 个“第二词元”的列表，这些列表是使用 SQL 命令自动汇总每个词元后生成的。最后，专家分析了完整列表（词元 + 相似度建议），从中挑选一项建议，或根据自身经验作出决定。整个过程如图 6 所示，下文也将对此进行举例说明。表 8 显示了汇总后，专家分两轮评估的 API 数量。

因此，他们没有对 Cronos 源代码中找到的 441 个 API 进行分类，而是检查了列表中第一词元和第二词元的自然语言处理建议。

表 8 每个项目中分类的 API 数量

项目	总计 APIs	APIs 已提交给专家 第一个词元	APIs 已提交给专家们 第 2 个词元	API % 第一轮	API % 第1 + 2名 轮次
Audacity	1,478	562	106	38.0%	45.1%
PowerToys	264	37	20	14.0%	21.6%
Rmca	8,645	10	95	1.7%	2.3%
Cronos	441	32	73	7.2%	23.8%
JabRef	1,692	137	45	8.0%	10.8%
总计 / 平均	12,520	869	339	6.9%	9.64%

该流程由三位专家（一位作者和两位高级开发人员）采用卡片分类法，手动接受或拒绝列表中每个词元对应的建议。每位专家根据自身经验，从建议中选取一个，或选择更合适的 API 领域。若专家不同意自然语言处理建议，也可以查看完整命名空间列表。例如，对于命名空间“com.oracle.xml.util.XMLUtil”中的第一个词元“xml”，相似度函数给出了可能的 API 领域标签及相似度值：输入和输出：0.7，错误处理：0.69，解析器：0.57。对于词元“util”，建议为：实用工具：0.9，数据结构：0.49。因此，命名空间“com.oracle.xml.util.XMLUtil”被分类为“实用工具”。大多数 API 都根据第一个或第二个词元进行分类。少数情况下（$< 10\%$），专家必须对完整命名空间进行分类。对所有词元完成分类后，专家又进行了一轮讨论以达成共识（所有项目共计 $\sim$16 小时）。

葡萄牙语项目采用了与英语项目相同的专家分类流程。事实上，Cronos 项目源代码中声明的库使用英语单词编写，因此不会影响自然语言处理分类。

我们将这 31 个类别（API 领域标签）用于此前收集的 22,231 个问题；这些问题是根据已更改文件中是否存在相应 API 来收集的。我们使用这个标注集构建多标签分类模型的训练集和测试集。

5.1.3 第*3*阶段 - 构建多标签分类 *ers*

由于解决一个问题可能需要多种类型的 API，因此我们采用了多标签分类方法。该方法已用于软件工程领域，例如对 Stack Overflow 中的问题进行分类（如 [18]），以及检测故障类型（如 [41]）和代码异味（如 [42]）。为了构建分类器，我们首先需要构建语料库，然后运行并评估这些分类器。

语料库构建。语料库构建包括预处理、清理、诊断，以及拆分为训练集和测试集。
预处理：我们构建了两个不同的模型——一个使用 TF-IDF [43]，另一个使用 BERT [44]。这些语料库包含所选问题的问题标题、正文和评论文本。

接下来，与其他研究类似 [43, 45, 46], ，我们采用 TF-IDF，这是一种通过为每个词分配权重来量化文档中词语重要性的技术。应用 TF-IDF 后，我们为每个问题中的词语获得了一组 TF-IDF 分数向量。向量长度取决于用于计算 TF-IDF 的词项数量，每个词项都会获得相应的 TF-IDF 分数。随后，我们将这些 TF-IDF 分数输入所选分类器之一（例如随机森林），以为每个问题添加标签。每个标签取二进制值（0 或 1），表示问题中是否存在相应的 API 领域。

对于 BERT，我们创建了两个独立的 CSV 文件：一个输入文件，其中包含与问题语料库配对的专家标注的 API 领域标签；另一个文件则列出特定项目可能使用的标签。BERT 无需额外的分类器，即可根据语料文本和标签列表直接为问题添加标签。

我们还通过将所有使用英语词汇的项目合并到一个数据集中，评估了分类器的性能。因此，我们还必须为所有项目构建一个新的组合ID（ID + 项目名称），以确保唯一性。在本次实验中，创建新 ID 后，我们合并了项目的二进制文件，包括每个项目中缺失的类（例如，RTTS 没有计算机图形学标签）。我们比较了各种算法，以确定最佳设置。

清理： 为了使用 TF-IDF 构建分类模型，我们将语料库中的每个单词转换为小写，并删除了 URL、源代码、数字和标点符号。我们还使用 Python nltk 软件包删除了停用词，并对单词进行了词干提取。我们筛除了问题和拉取请求模板[8]，因为它们重复的结构会引入噪声，而且在各个问题中并未得到一致使用。

我们遵循Izadi等人 [47] 的工作来处理BERT的数据。我们使用经过清理和未经清理的语料库对BERT进行了测试。我们通过比较使用经过清理和未经清理的语料库进行训练所得到的F值、精确率和召回率结果来进行验证。我们使用10折ShuffleSplit交叉验证进行了三次训练试验，并确定未经清理的语料库始终比任何清理方法（词干提取、移除停用词等）都能取得更高的指标。该结果与Izadi等人 [47] 的研究一致，他们表明，未经清理的输入语料库能够最好地保留BERT判断词语含义和重要性所需的词语上下文。

诊断： 多标签数据集通常用标签基数和标签密度来描述 [48]。标签基数是每个样本的平均标签数。标签密度是每个样本的标签数除以标签总数后，再对所有样本取平均值。对于我们的数据集，标签基数为 8.19，标签密度为 0.26。这些数值考虑了 22,231

[8] http://bit.ly/NewToOSS

经过上一节预处理步骤后获得的不同问题和API 领域标签。由于我们的密度可以被视为较高，多标签学习过程或推理能力不会受到影响 [49]。

训练集/测试集：我们使用 ShuffleSplit 方法 [48], 将数据拆分为训练集和测试集。该方法是一种模型选择技术，可为多标签分类器模拟交叉验证。例如，在 JabRef 项目中，我们有 1,914 个关联问题。由于一个 PR 可能关联多个问题，因此我们保留了 1,648 条记录，并随机将其拆分为训练集（分别包含 80%（1,318 个）、70%（1,154 个）和 60%（989 个）的问题）以及测试集（分别包含剩余的 20%（330 个问题）、30%（494 个问题）和 40%（659 个问题））。我们使用十组不同的训练集和测试集，按照 10 折交叉验证的方式运行每项实验十次。为改善数据集的平衡性，我们针对多标签方法运行了 SMOTE 算法 [50]。

分类器。为创建分类模型，我们选择了六种适用于多标签方法的分类器，并采用不同策略构建学习模型：决策树、随机森林（集成分类器）、MLPC 分类器（多层感知机神经网络）、MLkNN（基于传统 K 近邻算法的多标签惰性学习方法） [51, 48], 逻辑回归和 BERT。我们使用 Python sklearn 包运行前五种分类器，并测试了多个参数。对于表现最佳的分类器 RandomForestClassifier，我们采用了以下参数：准则 $=$ *'entropy'*，最大深度 $= 50$，叶节点最小样本数 $= 1$，拆分所需的最小样本数$=3$，估计器数量 $= 50$。

BERT模型使用开源 Python 软件包 Fast- Bert [52], 构建，该软件包基于用于 Pytorch 的 Transformers [53] 库。在训练模型之前，使用 LA MB 优化器 [54]计算最佳学习率。最后，模型经过 11 个 epoch 的拟合，并在每个 epoch 进行验证。这种训练和验证过程在ShuffleSplit10 折交叉验证的每个折中都会进行。BERT 模型在计算集群中配备的 NVIDIA Tesla V100 GPU 上进行训练。只要目标 GPU 具有足够的 VRAM 来训练 BERT 模型，硬件的选择就并不重要。

分类器评估：为了评估分类器，我们采用了以下指标（同样使用 scikit-learn 软件包计算）：

– **汉明损失** 衡量错误标签占标签总数的比例。– **精确率** 衡量正确预测的标签数量与预测标签总数的比例。– **召回率** 表示所有相关标签中被正确预测的标签所占的百分比。– **F值** 计算精确率和召回率的调和平均值。F值是一种加权指标，用于衡量预测出的相关标签数量，以及预测出的标签中相关标签的数量。

$$Precision = \frac{TP}{TP + FP} \tag{i}$$

$$Recall = \frac{TP}{TP + FN} \tag{ii}$$

$$FMeasure = \frac{2TP}{2TP + FP + FN} \tag{iii}$$

传统上，基于 TP、TN、FP和FN（真正例、真负例、假阳性和假阴性）计算精确率 (i)、召回率 (ii) 和 F值 (iii) 的经典公式解决的是单标签问题。在单标签问题中，一个实例被视为正确或不正确；而在多标签评估中，一个实例可能是部分正确的；也就是说，对于某些实例而言，只有部分类是正确的。为了解决多标签分类问题，相关文献 [55]建议对上述指标进行如下调整。

每个标签的指标可采用不同的平均策略计算，具体如以下公式所示。设 TP_l、 FP_l、 TN_l和 FN_l 分别为二元评估工作量 B（TP、TN、FP、FN）所返回的真正例、假正例、真负例和假负例数量，例如针对标签 l [56]进行二元相关性转换； q 为标签数量。宏平均 [55] 是所有逐标签指标的算术平均值，而微平均 [55] 是通过汇总 TP、FN 和 FP 得出的全局平均指标。计算这些平均值后，可用于计算精确率、召回率和 F值（i、ii、iii）。Santos 等人 [14] 使用微平均来计算预测结果的指标。因此，为了与先前的研究进行比较，我们也采用了该方法。微平均更偏向样本数量最多的类 [57]。

$$B_{macro} = \frac{1}{q} \sum_{l=1}^{q} B(TP_l, FP_l, TN_l, FN_l) \tag{iv}$$

$$B_{micro} = B\left(\sum_{l=1}^{q} TP_l, \sum_{l=1}^{q} FP_l, \sum_{l=1}^{q} TN_l, \sum_{l=1}^{q} FN_l\right) \tag{v}$$

迁移学习。接下来，我们研究使用不同的数据集训练和测试模型时各项指标的表现。我们使用英语词汇，将四个项目组合起来，其中三个用于训练，一个用于测试。例如，我们使用 JabRef、PowerToys 和 Audacity 训练一个数据集，再使用 RTTS 项目进行测试。随后，我们将测试数据集替换为训练集中的一个数据集，直到完成所有可能的组合。

数据分析。我们使用上述评估指标，以及每次模型运行后记录的混淆矩阵来评估分类器。我们使用 Mann–Whitney U 检验比较分类器的指标，随后进行 Cliff's Delta 效应量检验。我们根据 Romano 等人 [58], 提供的阈值评估 Cliff's Delta 的大小： $|d|<0.147$ 为“可忽略”，$|d|<0.33$ 为“小”， $|d|<0.474$ 为“中”，其余为“大”。我们将 p 值 < 0.05 作为判断统计差异的界限。

在后续分析中，我们筛除了未出现过的 API 标签。“云”和“机器学习”没有出现在所挖掘的任何问题/PR 中，因此没有预测结果。我们还在为每个项目运行模型时，筛除了出现在超过 90% 行中的标签。这些标签可能会使预测产生偏差，因为分类器总是可以推荐它们。例如，对于 PowerToys，标签“自然语言处理”“网络”“数据库”“错误处理”“语言”“开发运维”“输入输出”“机器学习”“安全”“云”“事件处理”“计算机图形学”“多媒体”“线程”“大数据”和“地理信息系统”均未出现，因此被移除。标签“实用工具”也被移除，因为它超过了标签阈值（出现在数据集中超过 90% 的行中）。“实用工具”标签出现次数最多，共 699 次，其次是“应用程序”，出现 501 次，以及“用户界面”，出现 498 次。代表性较低的一组标签包括“测试”（6 次）、“日志记录”（6 次）和“国际化”（4 次）。PowerToys 是一组适用于 Microsoft Windows 的实用工具。“实用工具”标签出现频率很高，这在预期之中。

使用包含所有项目的数据集进行预测后，标签的分布发生了显著变化。出现频率最高的标签是“UI”，共出现 762 次，其次是“Util”，出现 726 次，以及“Logic”，出现 575 次。出现频率较低的标签包括：“NLP”（45 次）、“CG”（16 次）和“GIS”（10 次）。尽管一些标签很受欢迎，专家也曾用它们为许多 API 添加标签，但提交的拉取请求中很少有涉及包含这些 API 的源代码，这或许可以解释它们为何如此少见。提及这些标签的关联问题和拉取请求较少，也可能导致它们未出现在数据集中。最后，将所有数据集一起用于训练有助于拉开各标签的出现频率。例如，在训练 JabRef项目时，由于“Util”和“Logic”标签的预测结果达到 90% 的阈值，因此被舍弃。而使用合并了所有项目的数据集进行训练时，这些标签得以保留，低于 90% 的阈值，并被用于为问题添加标签（图 7）。

最后，我们检查了每个问题所包含的标签数量分布（图 8）。我们发现，有 110 个问题包含六个标签，106 个问题包含三个标签，104 个问题包含七个标签，102 个问题包含八个标签。只有 4.1%（$=40$）的问题包含一个标签，这证实了这是一个多标签分类问题（图 8）。

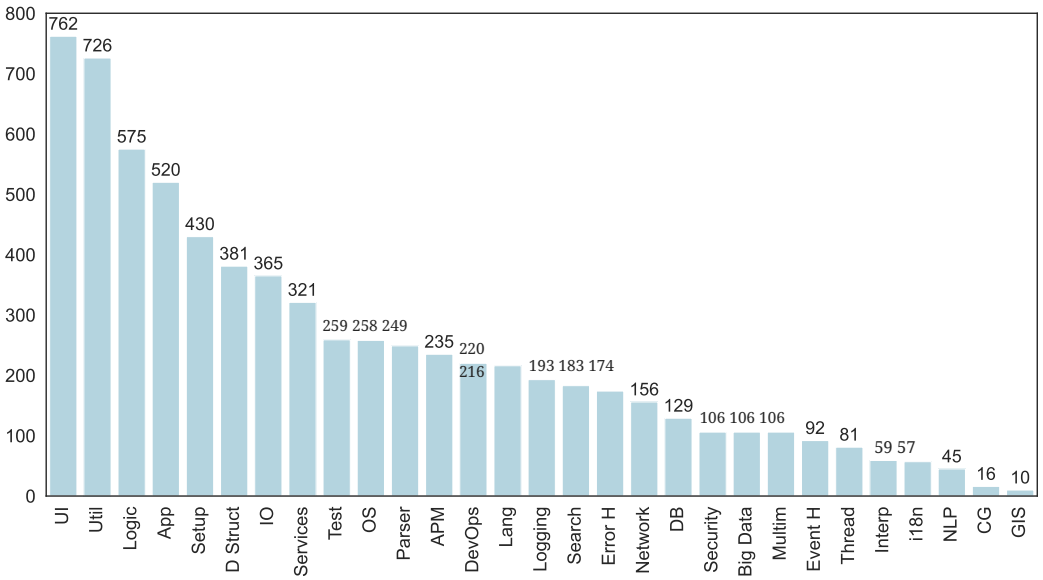


图 7 合并所有数据集的模型按类型统计的标签数量

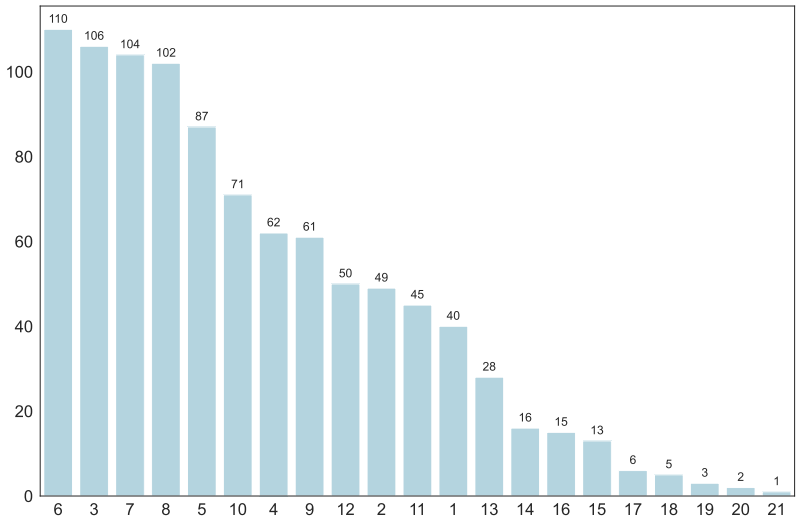


图 8 合并所有数据集的模型按问题统计的标签数量

5.2 结果

RQ.2.1: 我们在多大程度上可以自动分配 API 领域标签使用项目中的数据来处理问题?

为了预测API 领域标签，我们首先测试了一个简单的语料库：仅将问题 TITLE 作为输入，并使用随机森林（RF）算法，因为它对参数设置 [59] 不敏感，而且已在软件工程研究中显示出良好的预测结果 [60, 61, 62, 63]。然后，我们评估了语料库配置的各种替代方案，改变输入信息：仅 TI-TLE （T）、仅 BODY （B）、 TITLE 以及 BODY （T+B），以及 TITLE、 BODY和COMMENTS （T+B+C），并比较所有项目的平均值。为了比较不同的语料库配置，我们保留随机森林算法，并使用带有 Cliff's-delta 效应量的 Mann-Whitney U 检验。

我们还使用 n 元语法测试了替代配置。对于每个步骤，都保留了最佳配置。然后，我们使用了不同的机器学习算法，并将其与虚拟基线（随机）分类器进行了比较。

如图 9 和表17（附录 A）所示，当我们测试不同输入并将其与 TITLE o nly进行比较时，所有替代设置在使用 TF-IDF 时都取得了更好的结果。与先前的研究 [14]相比，我们在精确率、召回率和 F值方面均观察到了改进。使用 BODY 时，我们达到了 84% 的精确率、78.6% 的召回率和 81.1% 的 F值。相比之下，尽管 BERT 的结果较差，但使用 TITLE 的模型表现优于其他 BERT 模型，精确率达到 61.6%。

表 9 TF-IDF 和 BERT 的语料库模型替代方案在 F值和精确率方面的 Cliff's Delta 比较。标题（T）、正文（B）和评论（C）。

TF-IDF/BERT	**语料库比较**	**Cliff's Delta**			
		F值		**精确率**	
TF-IDF	T 与 B	-0.005	可忽略	-0.15	小***
TF-IDF	T 与 T+B	-0.10	可忽略***	-0.12	可忽略***
TF-IDF	T 与 T+B+C	-0.03	可忽略***	-0.01	可忽略
TF-IDF	B 与 T+B	0.10	可忽略***	0.02	可忽略
TF-IDF	B 与 T+B+C	-0.02	可忽略	0.14	可忽略***
TF-IDF	T+B 与 T+B+C	0.07	可忽略***	0.11	可忽略***
BERT	T 与 B	0.07	可忽略	0.11	可忽略
BERT	T 与 T+B	0.13	可忽略的	0.03	可忽略的
BERT	T 对比 T+B+C	0.03	可忽略的	0.09	可忽略不计的
BERT	B 对比 T+B	0.10	可忽略	-0.04	可忽略
BERT	B 与 T+B+C	-0.006	可忽略	0.08	可忽略
BERT	T+B 与 T+B+C	-0.09	微不足道	-0.01	微不足道

** $p \leq 0.05$; ** $p \leq 0.01$; *** $p \leq 0.001$*

对于 TF-IDF，将仅使用TITLE 与其他三种语料库配置的结果进行比较后，我们发现存在统计学差异：F值（与 TITLE+BODY 或 TITLE+BODY+COMMENTS比较时，p 值≤ 0.001，

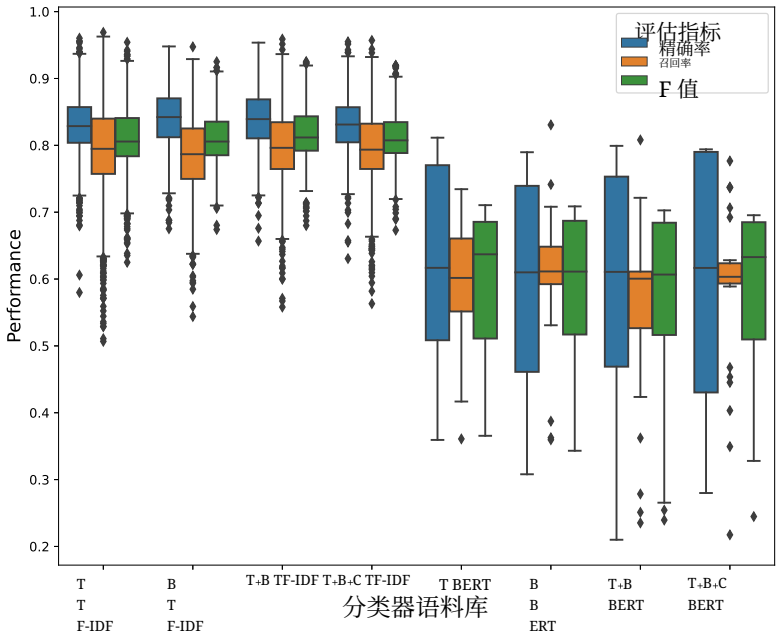


图 9 输入 TF-IDF 和 BERT 的语料库模型比较。T=标题，B=正文，C=评论

Mann-Whitney U 检验）和精确率（与BODY 或 TITLE+BODY比较时，p 值 ≤ 0.001，Mann-Whitney U 检验）；比较 TITLE 和 BODY的精确率时，两者的效应量均可忽略不计。采用 BODY 配置的语料库在精确率方面优于所有其他配置；紧随其后的是采用 TITLE+BODY配置的语料库，它在召回率和 F 值方面表现更好。不过，结果表明，仅使用 BODY就能取得足够好的结果，因为与其他两种配置相比，效应量可以忽略不计——在 BODY之外再加入 TITLE 和/或 COMMENTS，所取得的结果相近，但工作量更少。表 9 展示了各语料库配置两两之间的 Cliff's delta 比较，图 9 则通过箱线图证实，三种不同设置得出的结果相近。对于 BERT，所有模型在精确率和 F值上的分布都相同。

接下来，我们研究了二元语法、三元语法和四元语法的使用，并将结果与一元词的使用进行了比较。在此分析中，我们仅使用包含问题 BODY 的语料库，因为在前一步中选择了这一配置。表 18（附录A）和图10展示了算法在每种n元语法配置下的表现。虽然一元词配置的 F值略高，但四元语法的精确率略高。不过，它们在精确率上的差异具有可忽略的效应量，而它们在 F 值上的差异具有较小的效应量。此外，一元词

使用更少的计算工作量和内存 [64]。因此，我们保留一元词作为最佳选项。

表10 Cliff's Delta：F值和精确率的比较：n 元语法模型之间的比较

n 元语法比较	Cliff's Delta			
	F值		精确率	
1 对 2	0.09	可忽略的	-0.02	可以忽略**
1 对 3	0.11	可忽略***	-0.01	可忽略的
1 对 4	0.15	小***	-0.06	可忽略
2 与 3	0.02	可忽略	0.01	可忽略***
2 与 4	0.06	可忽略***	-0.04	可忽略**
3 与 4	0.04	可忽略**	-0.05	可忽略***

$* \ p \leq 0.05; \ ** \ p \leq 0.01; \ *** \ p \leq 0.001$

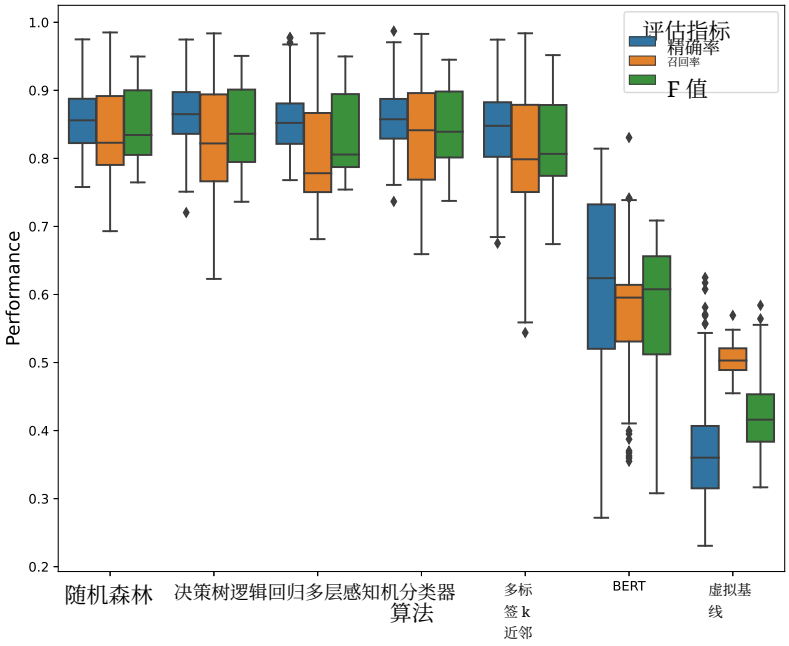


图 10 机器学习算法的性能比较

为研究机器学习（ML）分类器的影响，我们以 BODY 一元词作为语料库，对多个选项进行了比较。这些选项包括：随机森林（RF）、神经网络多层感知器（MLPC）、决策树（DT）、LR、MlKNN、BERT，以及策略为“uniform”的虚拟基线分类器。虚拟基线或随机分类器通常被用作

表11 Cliff's Delta 的 F值和精确率：机器学习算法之间的比较

算法比较	Cliff's Delta			
	F值		精确率	
随机森林与逻辑回归	0.27	small***	0.06	可忽略的*
随机森林与 MLPC	0.02	可忽略的	0.009	可忽略的
随机森林与决策树对比	0.06	可忽略*	0.09	可忽略***
随机森林与多标签 k 近邻对比	0.28	小***	0.13	可忽略的***
随机森林与 BERT	1.0	large***	1.0	大***
逻辑回归与 MLPC	-0.21	小***	-0.07	可忽略*
逻辑回归与决策树	-0.15	小***	-0.15	小***
逻辑回归与多标签 k 近邻	0.07	可忽略*	0.08	可忽略*
逻辑回归与 BERT	1.0	大***	1.0	大***
MPLC 与决策树	0.03	可忽略的	-0.08	可忽略***
MPLC 与多标签 k 近邻	0.24	小***	0.13	可忽略***
MLPC 与 BERT	1.0	大***	1.0	large***
多标签 k 近邻与决策树的比较	-0.19	小***	-0.20	small***
多标签 k 近邻与 BERT***	1.0	大	1.0	大型***
决策树与 BERT 对比***	1.0	大型	1.0	大型***
随机森林与虚拟基线对比	1.0	大型***	0.50	large***

** $p \leq 0.05$; ** $p \leq 0.01$; *** $p \leq 0.001$*

基线 [65, 66]。我们使用了 Python 软件包 scikit- learn[9]中的实现。图10和表19（附录 A）展示了这些算法之间的比较，表11则给出了使用 Cliff's Delta 对 F值和精确率进行两两比较的统计结果。

与决策树（DT）、逻辑回归（LR）、神经网络多层感知器（MLPC）、多标签 k 近邻（MlKNN）算法以及 BERT 相比，随机森林（RF）是表现最佳的模型。在考虑 F 值和精确率时，随机森林的表现优于这五种算法，效应量为可忽略或较小。与 BERT 和虚拟基线分类器相比，效应量较大。部分算法之间观察到的差异相当小，因此可能会因项目语料库的属性而有所不同。

结果表明，该分类器适用于预测以不同编程语言（C++、C# 和 Java）编写的项目中的标签，且问题的词汇使用英语和葡萄牙语。

> ***RQ.2.1*** 总结。使用随机森林算法、 BODY 作为语料库以及一元词，可以为每个项目单独预测 API 领域标签，精确率为 0.864，召回率为 0.786，F 值为 0.811。

RQ.2.2：我们可以在多大程度上利用其他项目的数据，自动为问题分配 API 领域标签?

接下来，我们合并了使用英语词汇的数据集（RTTS、JabRef、Audacity 和 PowerToys），为所有项目预测 API 领域标签。

[9] `https://scikit-learn.org/`

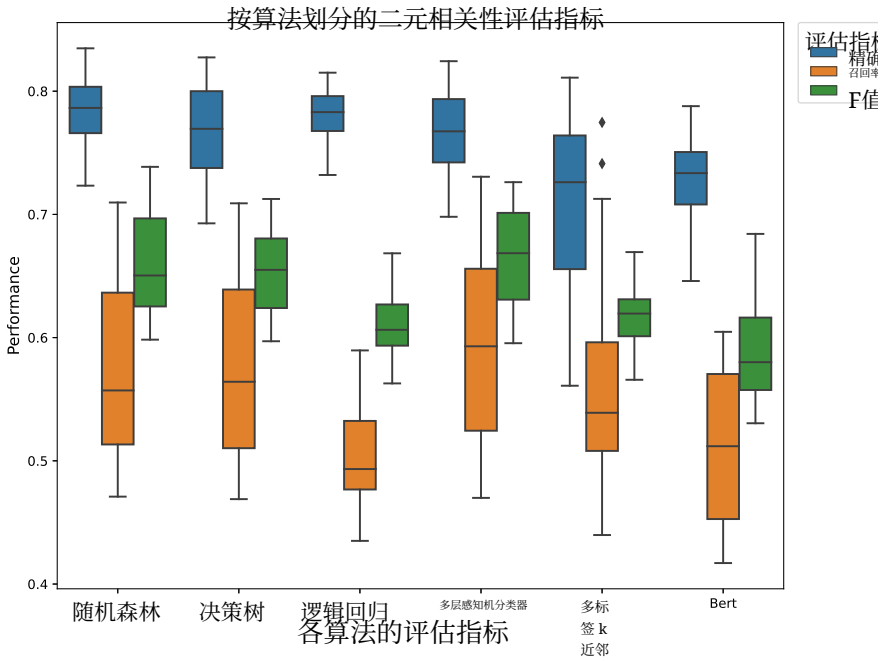


图11 使用包含所有项目的数据集比较机器学习算法的性能 -词汇表：EN

由于BERT模型使用英语词汇进行训练，因此有必要移除使用葡萄牙语词汇的项目。预测结果以 BODY 作为语料库进行（TF-IDF 使用一元词）。图11展示了使用不同算法获得的性能。RF仍具有最佳精确率，而MLPC具有最佳F值；BERT 的精确率优于多标签 k近邻，召回率优于逻辑回归。在使用所有项目合并数据集预测 API 领域标签时，BERT 受指标损失的影响较小（表20 - 附录 A）。

研究问题***2.2***总结。 使用合并所有英语项目的数据集进行预测，使随机森林的精确率降低了 9.15%，而使用 BERT 的精确率提高了 20.63%。

研究问题***2.3:*** 我们在多大程度上能够使用迁移学习自动为问题标注 API领域标签?

最后，表12展示了使用迁移学习测试的所有组合的结果。结果的精确率和召回率差异显著：预测 RTTS 和 PowerToys 时，精确率分别为 0.713 和 0.296；预测 Audacity 和 RTTS 时，召回率分别为 0.525 和 0.175。

表12 用于评估迁移学习的模型总体性能。

训练	Test	P	R	F
RTTS、Audacity、PowerToys	JabRef	0.305	0.294	0.299
JabRef、Audacity、PowerToys	RTTS	0.713	0.175	0.281
JabRef、RTTS、PowerToys	Audacity	0.688	0.284	0.402
JabRef, RTTS, Audacity	PowerToys	0.296	0.525	0.379
JabRef、Audacity、PowerToys	RTTS*	0.718	0.272	0.394

** RTTS - labels most important to the users (Section 6.2)*

此外，我们还开展了一项迁移学习实验，目标是针对由开发者评估的RTTS项目标签（第 6.2 节）。我们剔除了评估次数少于三次的所有标签，以及最多50% 的“ 不重要”评估（见图12）。因此，在 RTTS 项目中保留下来的标签为：“网络”、“日志记录”、“设置”、“微服务”和“用户界面”。由于 Audacity、JabRef和PowerToys 项目未经过开发者评估（第6 节），因此未纳入此次实验。我们观察到精确率小幅提升（从 0.713提升至 0.718），召回率（9.7%- 0.175提升至0.272）和 F值（11.3% - 0.281 提升至 0.394）显著提升——表12。

研究问题***2.3***总结。 考虑源项目和目标项目的多种配置进行迁移学习，使指标下降了15.12%至64.74%，召回率下降了33.21%至77.74%，具体取决于源项目和目标项目。仅针对解决问题的开发者认为重要的API领域标签评估迁移学习，使召回率提高了9.7%，F值提高了11.3%。

6 RQ3 - 与开发者一起评估 API 领域标签

鉴于人类输入在机器学习研究中非常重要，我们标记了一些问题，并将其呈现给此前解决过相同问题的开发者，以获取他们的反馈，了解如果在他们处理这些问题时能够使用 API领域标签，这些标签会有多大用处。

6.1 方法

为回答 RQ3，我们使用随机森林算法，以问题描述 BODY作为语料库，并使用一元词（我们在 RQ.2 中找到的最佳配置）为问题生成标签。

6.1.1 标签生成

我们为 91 个问题预测了标签（PowerToys $= 21$、Audacity $= 18$、Cronos $=$24 和 RTTS $= 28$）。预测结果涵盖了全部 29 个拟议的 API 领域标签（我们的项目中没有云和机器学习的样本）。我们选择了 the

从项目中最近关闭的问题入手，以更有可能找到修复问题的开发者，并让他们回忆起问题已得到解决。然而，当解决问题的贡献者已经不再为该企业工作，或开源软件贡献者没有回复我们的联系时（第 6.1.2节），我们不得不舍弃一些问题。使用最近的问题以及参与者的可用性，导致用于评估的标签集合不平衡；在实证实验中，我们尽最大努力纳入尽可能具有代表性的 API 领域标签集合。

6.1.2 贡献者评估

在这一步中，我们招募了 20 名参与者（PowerToys（1 人）、Cronos（13 人）和 RTTS（6 人））。为了从这些项目中招募参与者，我们向 PowerToys 和 Audacity 的维护者发送了电子邮件，并联系了 Cronos 和 RTTS 的开发经理。我们请这些项目的参与者评估这些标签是否体现了解决问题所需的技能，以及是否能帮助希望选择问题的新手或有经验的开发者。所有参与者都是各自项目的专家，并受邀评估自己过去曾参与过的问题。实际上，参与者评估的问题数量因其过往贡献而异。每个问题仅由一名参与者评估。参与者获赠礼品卡，以感谢他们参与研究。

我们提出了以下问题：

– 您认为在问题上添加这些标签，对于帮助新贡献者识别解决问题所需的技能有多重要？（请评估每个标签）（李克特量表：非常重要、重要、中等重要、稍微重要、不重要）– 为什么？– 还缺少哪些标签？

6.1.3 分析

基于从贡献者评估中收集的数据，我们进行了定量分析，以评估生成的标签。为了分析贡献者可以解释其对所生成标签看法的开放式问题，我们采用了开放式编码和主轴编码流程 [67]。

6.2 结果

在预测的91 个问题中，我们收到了 29.67% 的反馈（26 个问题和 16 个不同的 API 领域标签）。我们没有收到 Audacity 贡献者的反馈。PowerToys 只有一位贡献者，仅评估了3 个问题，涵盖 4 个标签。由于数据不足，我们从结果中移除了该项目。

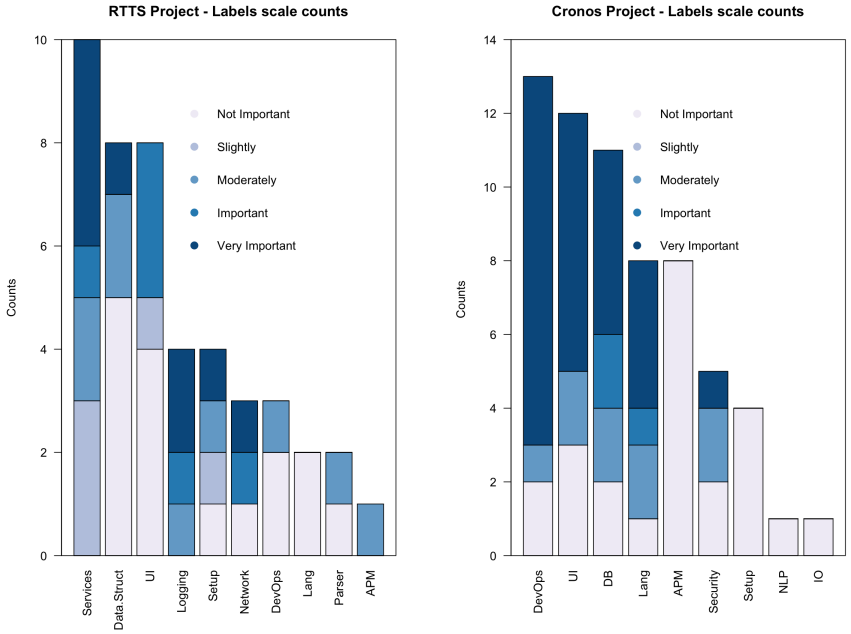


图 12 按项目进行的标签评估

Cronos。共有13 位贡献者评估了生成的标签。根据结果（图12），贡献者认为5 个标签（即开发运维、用户界面、数据库、语言和安全）非常重要或重要，并认为应用性能管理、设置、自然语言处理和输入输出标签不重要。开发运维、用户界面、数据库和语言被高度评价为重要，其中许多评价为“非常重要”和“重要”。并非所有参与者都说明了理由，但在贡献者提到的理由中，有人指出“这是一个简单的用户界面问题。”（P15），这表明“用户界面”预测取得了成功。另一位开发者提到：“这个问题还需要数据库、逻辑和语言技能。”（P6）。该问题被标记为“用户界面”和“开发运维”（评价为“非常重要”），但开发者遗漏了一些技能。关于缺失的标签，一些贡献者提到：“这个问题与限制有关。它需要用户界面技能和开发运维技能（预测结果中未包含这些技能）。 *[.....]*但它还需要其他技能。”（P18）。另一位贡献者也遗漏了一些标签，并提到：“这个问题还需要数据库、逻辑和语言技能。”（P06）。

RTTS。 关于RTTS项目，五位贡献者评估了他们所解决的问题所生成的标签；在这一场景中，每位贡献者评估了 1到3 个问题。我们的发现（图12）突出显示，以下标签被贡献者归类为重要到非常重要：服务、用户界面、日志记录、设置、网络和数据结构。在贡献者强调的原因中，我们可以看到如下所述的积极反馈：“对于这一点的标签，我完全同意，因为这样可以找到问题

并应用解决方案，所有技能都是必要的。”(P20)。此外，贡献者将以下标签归类为不重要或略微重要：语言、解析器、开发运维、用户界面和数据结构。一些贡献者提到：“我部分同意，其中一些标签可以为所报告的问题提供初步视角，但有些并不相关，例如语言、数据结构和设置” (P2)。一些贡献者根据以下提及的内容指出缺少某些标签：“要排查问题并从应用程序中获取相关信息，日志记录技能是必要的；而服务技能（了解系统中的服务发现和服务注册表如何运作）对于发现所请求服务的版本与已注册的版本不匹配也是必要的。至于网络，能够确定该问题并非由所请求服务返回的某些错误或故障响应导致，可能会有所帮助，但在本例中，日志明确说明所请求的服务不存在。我认为其他标签/技能并不适用于这个问题。” (P04)。

RQ.3. 总结。 我们的研究结果表明，这些标签有助于识别解决问题所需的技能。不同项目中生成的标签效率有所不同：Cronos 项目中，61.9% 的标签被评为从略微重要到非常重要；RTTS 项目中，这一比例为 64.4%。

7 讨论

本节讨论我们的结果和未来工作。

开发者是否对标签有明确的偏好？ 第4节中研究参与者分享的反馈向我们展示了不同类型的标签对于简化问题选择过程的重要性。然而，开发者表达了对不同类型标签的偏好，而其中一些偏好并不明确。例如，当我们询问他们希望在问题中看到哪种类型的标签时，P32表示“所使用的技术”。这项技术可能指“编程语言”或“API”。虽然这两种分类都可以使用，但我们更倾向于将该技术定义为 API，因为与编程语言，甚至是能够涵盖许多库的框架相比，API更为具体。类似的情况也出现在“优- 先级”（P12）上。“优先级”可能仅限于“低”或“高”，也可能包括其他方面，例如 P15 所建议的“对运营的影响”。

我们可以将技术领域（技术、API、程序语言、数据库、框架和架构层）以及管理领域（类型、优先级、状态、难度级别、GitHub信息）中的标签类型进行分组。管理标签更常用于问题跟踪器中。在这项工作中，我们提议为问题添加一种技术标签，即 API 领域标签；我们认为，这些标签可以作为解决问题所需技能的代理指标。然而，应避免用过多标签给问题跟踪器增加负担。未来研究可以进-

在为贡献者提供标签与避免造成视觉负担之间寻求恰当的平衡。

API 领域标签是否相关? 我们的研究结果表明，参与者认为 API 领域标签与问题的选择相关。更具体地说，项目的新手认为 API 领域标签比描述组件的其他通用标签更相关，也略微比描述问题类型的管理标签更受青睐。这表明，对 API 领域的高层次理解比对项目中特定组件的更深入信息更相关。

在控制问题类型和组件后，与新手（或学生）相比，API 领域标签被认为与有经验的程序员更相关。这表明，新手可能需要的不只是他们需要掌握技能所涉及的技术方面的帮助。我们的结果还表明，如果问题中能提供更多信息，例如复杂度级别、所需的特定 API 知识、达到相应技能水平所必需或推荐的学术课程、预计完成时间、求助联系人等，就能为新手提供帮助。

尽管每位贡献者在转到新项目时都是新手，但当新项目与之前的项目共享技术时，先前的经验仍然很重要。与经验丰富的新手不同，后者可能会从之前的项目中迁移知识，并直接着手解决问题；新手新成员则会花费更多时间来了解项目结构、底层技术以及如何设置环境 [11] 这或许可以解释为什么来自业界的从业者和经验丰富的参与者，比学生和新手选择了更多的 API 领域标签。也许 API粒度比新手所寻找的更深。未来的研究可以考虑为新手新成员提供适当的技术信息。

除了 API 领域标签之外，还有哪些问题特征与识别问题中的技能相关? 除了标签之外，新贡献者还提到了 TITLE、 BODY 和 COMMENTS ，认为它们是识别处理问题所需技能的信息来源。这些要素可以通过问题模板进行结构化，也可以采用临时方式加以说明。Santos 等人 [11] 要求维护者提出社区策略，以帮助新手找到合适的问题。在已确定的策略中，维护者建议了 15 种为问题添加标签的不同方式（例如使用技能、知识领域、编程语言、库等进行标注），以及多种组织问题的方式，其中包括创建模板。

虽然这些其他问题要素可能表明问题所需的技能以及标签未涵盖的其他特征，但有些问题和现有模板并不完整，缺少对贡献者很重要的信息。本文采用的 5W2H 分析可以帮助我们全面理解问题中应写明哪些内容，涵盖信息的七个维度：谁、为什么、何时、什么、何地、如何解决以及问题有多大。未来的工作可以使用 5W2H 问题，检查现有模板是否完整涵盖这七个信息维度。

尽管问题模板很重要，但为了清理将被数据处理流水线摄取的重复文本，我们删除了模板句子。例如，其中一个模板句子是“复现步骤”。由于这段固定文本出现在许多问题中（无论它们属于何种类别），并且模板随着时间的推移发生了变化，我们决定在处理问题语料库之前将其删除。这一删除仅影响训练好的模型，而我们仍应使用5W2H 分析的结果来创建一个面向人类的模板，以便为新贡献者指引与其相关的信息。

语料库特征对标签分类有何影响? 通过观察将不同语料库作为输入时报告的结果（TF-IDF），我们注意到，仅使用问题正文创建的模型，其性能与使用问题标题、正文和评论的模型相近，并且优于仅使用标题的模型。通过检查结果，我们发现，添加更多单词来创建模型后，特征矩阵会变得稀疏，并不会提升分类器的性能。

我们还发现了标签之间的共现。例如，“测试”“日志记录”和“国际化”经常同时出现（图 13）。这是由于源文件中存在密切关系。通过在源代码中搜索这些 API 领域类别的引用，我们发现“测试”出现在 4,579 个源代码文件中，而“日志记录”出现在 903 个文件中。标签“国际化”仅出现在 73 个文件中。另一方面，“计算机图形学”和“安全”的 API 领域标签通常不会共现。“计算机图形学”仅出现在 5 个 Java 文件中，而“安全”仅出现在 47 个文件中。未来的研究可以探讨共现技术，以预测此上下文中软件工件（例如， [68]）的共同变更。

图13 展示了包含所有项目的数据集中的标签共现情况。共现矩阵呈现了每个标签与其他所有可能标签出现在相同上下文中的次数。通过检查上述共现数据，我们可以确定一些预期情况并得出一些预测结果。例如，“DB”标签（数据库）更频繁地与“网络”和“线程”同时出现。因此，当一个问题同时具有这两个标签时，我们可以进行猜测；即使机器学习算法无法预测出“数据库”标签，我们也很可能可以建议添加该标签。一项可能的未来工作是将本文提出的机器学习算法与频繁项集挖掘技术（如 Apriori）相结合 [69]。

准确标注有哪些困难? 我们怀疑，与“i18n”、“解释器”、“地理信息系统”和“自然语言处理”标签的低出现率（$<$ 57个问题）相比，“用户界面”、“实用工具”和“逻辑”标签的高出现率（$>$ 500 个问题）可能会影响精确率和F值。我们仅使用出现频率最高的前5个 API 领域标签对分类器进行了测试，未发现具有统计显著性的差异。一种可能的解释是，用于创建分类器的转换方法是二元相关性，该方法会为每个标签创建一个独立的分类器，而忽略潜在的共现。

由于项目的特性，数据集是不平衡的。例如，JabRef是一个桌面应用程序，因此 API领域标签“用户界面”出现得更频繁。表 13 展示了包含所有项目的数据集的混淆矩阵（单个项目请参见附录）。这一-

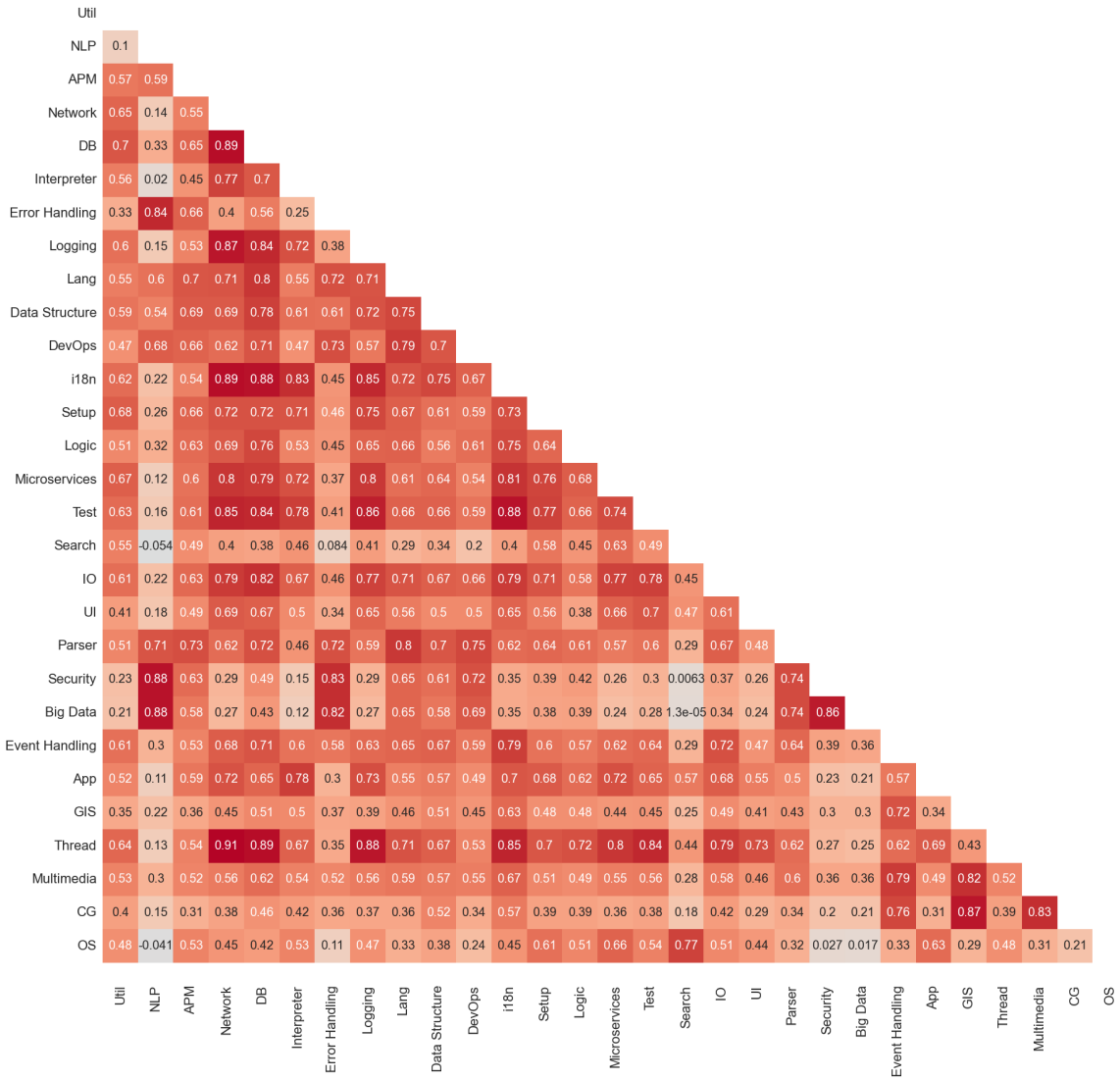


图 13 热力图 - 合并所有项目后的数据集中的标签相关性。颜色越深，标签之间的相关性越强。

即使使用 SMOTE 算法来增加稀有标签的出现次数，对少数类标签的预测仍会受到影响。有些标签只出现在少数几个项目中。因此，即使在某个特定项目中很常见，当使用所有项目进行训练和测试时，这些标签也可能变得稀有。应避免推荐结果不佳的标签，因为这可能会让贡献者掌握错误技能。

表 13 使用所有项目时所选模型的混淆矩阵数据和性能

API领域	TN	FP	FN	TP	精确率	召回率
APM	125	4	44	8	0.66	0.15
App	80	22	19	60	0.73	0.75
大数据	152	0	29	0	0	0
数据结构	78	24	6	73	0.75	0.92
DB	163	2	11	5	0.71	0.31
开发运维	113	26	0	42	0.61	1
错误处理	97	32	5	47	0.59	0.90
事件处理	162	1	8	10	0.9	0.55
GIS	178	2	0	1	0.33	1
解释器	173	2	3	3	0.6	0.5
IO	141	8	5	27	0.77	0.84
i18n	166	7	5	3	0.3	0.375
Lang	112	36	0	33	0.47	1
日志记录	174	1	4	2	0.66	0.33
逻辑	68	9	2	102	0.91	0.98
微服务	151	1	23	6	0.85	0.2
网络	175	0	6	0	0	0
NLP	164	0	17	0	0	0
OS	119	9	8	45	0.83	0.84
解析器	101	28	3	49	0.63	0.94
搜索	134	9	15	23	0.71	0.6
安全	151	0	30	0	0	0
设置	38	56	9	78	0.58	0.89
Test	166	0	15	0	0	0
UI	10	33	3	135	0.8	0.97
Util	84	4	16	77	0.95	0.82
总计	3275	316	286	829		

尽管在预测稀有标签方面的准确率不高，但对于出现次数超过200次的标签（所有项目合计），我们仍能够以合理的精确率（0.84）和/或召回率（0.78）进行预测。我们认为，项目的性质会影响与其领域相关的问题数量。例如，由于 Audacity项目是一个音频编辑器和录音机，因此“用户界面”、“输入输出”和“多媒体”标签出现频率较高是可以预期的。我们认为，Audacity的性质促成了与上述标签相关的问题数量。样本较少的标签的指标较低或不稳定。例如，根据文本/训练集划分的不同，“数据库”的预测召回率在 0.09 到 0.9 之间变化。

提升 BERT 的性能。除了标签的出现次数之外，还可以通过增加

训练集大小。Wang 等人 [29] 对多个经过训练的深度学习模型用于 GitHub 标注的探索，为使用 BERT 可能带来的性能提升提供了重要见解。作者表明，对于至少包含 5,000 个问题的大型数据集，BERT 模型的表现优于其他语言模型，在准确率、精确率、召回率和 F值方面均取得了最高分数。然而，对于少于 5,000 个问题的小型数据集，卷积神经网络作为整体最佳模型，表现优于 BERT。这表明 BERT 依赖于语料数据训练集的大小。因此，当目标开源项目的数据集大小增加时，BERT 在标注 GitHub 问题方面的性能将得到提升。当项目数据集合并后（表 11），与其他分类器相比，BERT 指标在精确率方面的差距从约 26% 降至 6%。

专家分类的影响是什么？ 专家还可以帮助提高所有模型的分类指标。我们可以观察到，C++ 项目与 Java 和 C#项目相比取得了最佳的F值（分别为 0.84、0.82和0.80，效应量从小到大不等）。尽管我们只评估了一个 C++项目，但在检查表8 后，结果可能表明，专家评估的 API 数量会影响我们将获得的指标。另一方面，手动评估大量 API可能会使泛化变得不可行。专家对 C++ 项目开展的分类涵盖了更高百分比的已分析 API。这可能是由语言特征造成的：从 C++ 源代码中解析出的库名称包含的有关其用途的信息有限。因此，分类耗时更长。事实上，C++ 项目需要专家投入更多工作量来进行分类。最终，这成为一次更详细的分类，并获得了更好的预测指标。

尽管专家分析耗时较长，但有些离群项目所需的工作量远少于其他项目。例如，专家分析的 API 不到 RTTS 中 API 总数的 3%。由于该项目导入了热门库，在整个源代码中复用了许多库，并且采用了模块化设计，因此专家的工作更加轻松。未来工作应进一步探索热门 API、模块化与专家评估之间可能存在的关系。另一项可能的未来工作是确定哪些编程语言特征会影响专家分类。

所提出的方法在多大程度上具有普适性？ 半自动分类流程减少了专家判定 API 专业领域所需的工作量。尽管仍需投入相当多的工作量，但随着数据集规模扩大，需要分类的新 API 的增长率应会下降，因为项目平均会复用 35-53% 的核心 API。第三方库占比为 8-32%，平均为 45%（核心 +第三方）。使用热门开源 API，可能会使不同项目之间共享的 API 高达 85% [70]。此外，项目规模的增长速度远快于仅在特定项目中使用的 API 实体数量的增长速度 [70]。

因此，当挖掘出的库数量达到临界规模（针对每种编程语言）时，对专家评估的需求应显著下降，甚至新项目也可以使用以往的专家评估。此

可能会影响应用于工业项目的方法，这些项目可能使用各种独特的专有 API。然而，API 共享可能发生在公司或业务部门内部，从而重复库的临界规模现象。尽管如此，我们没有观察到这种影响，并且可以使用来自工业项目的数据预测开源软件项目的标签。

模型在多大程度上执行迁移学习？当项目缺少训练数据（冷启动），或缺少开发自有模型所需的时间或基础设施时，迁移学习至关重要。在业界，这个问题可能尤为突出，因为出于安全措施或遵守程序及法律的要求，数据的访问可能受到限制。在这种情况下，使用预训练模型的能力必不可少。使用预训练模型对新数据进行预测也很有优势，因为每当数据集中新增一个源项目时，这种做法都比重新训练模型更快、成本更低 [15,71]。当项目的数据集太小时，项目也可能受益于其他项目的补充数据，以训练预测模型。

迁移学习实验发现，精确率和召回率均有所下降。指标定义：$Precision = \frac{TP}{TP+FP}$和 $Recall = \frac{TP}{TP+FN}$表明假阴性（FN）和假阳性（FP）的数量会对结果产生影响。例如，在对各个项目分别进行训练和测试时，与将 RTTS 项目作为目标项目的迁移学习实验相比，RTTS 项目的假阳性和假阴性数量较少（表14和表15）。以 RTTS 项目为目标时，假阴性数量较多，显著降低了召回率。另一方面，以 PowerToys 为目标时，假阳性数量对精确率产生了负面影响（表15、表22和表12）。这些项目仅共享少量标签（31 个中有 5 个），且各数据集之间类别分布不均衡。例如，“设置”在 RTTS 项目中很常见，在 JabRef 中却很少见，这表明源项目与目标项目的条件概率分布存在差异。这些特征可能决定哪些项目彼此匹配，因此可用于选择迁移学习的源项目或目标项目。未来的研究应探讨领域、平台（Web、桌面、移动端）、架构或其他项目属性是否有助于找到良好的匹配。此外，还可研究代理技术，例如 Nam 等人提出的方法 [15]，，通过缩小目标项目与源项目之间的数据分布差异来预测软件工程缺陷，并将其应用于 API 领域标签的预测。附录 A 中列出了使用迁移学习得到的 JabRef（表25）和 Audacity（表26）项目的结果。对比表26中 Audacity 的迁移学习结果与单独对 Audacity 数据集进行训练和测试的结果（表23），可以看到假阳性和假阴性的数量较多。同样，对比表21和表25中的 JabRef 结果，也可以观察到相同的模式。

贡献者如何评价为他们解决的问题生成的标签？

总体而言，参与者对生成的标签给予了积极反馈。在所有项目中被归类为重要或非常重要的标签有：开发运维（10）、数据库（5）、服务（4）、用户界面（14）、语言（5）、安全（1），以及

表 14 混淆矩阵和性能。仅使用 RTTS 项目进行训练和测试

API领域	TN	FP	FN	TP	精确率	召回率	F值
APM	112	4	2	24	0.85	0.92	0.88
大数据	134	1	2	5	0.83	0.71	0.76
数据结构	25	36	3	78	0.68	0.96	0.80
DB	84	4	19	35	0.89	0.64	0.75
开发运维	60	17	13	52	0.75	0.80	0.77
错误处理	126	5	4	7	0.58	0.63	0.60
事件处理	129	0	2	11	1	0.84	0.91
i18n	121	6	7	8	0.57	0.53	0.55
Lang	28	29	11	74	0.71	0.87	0.78
日志记录	47	24	18	53	0.68	0.74	0.71
微服务	1	12	0	129	0.91	1	0.95
网络	65	20	21	36	0.64	0.63	0.63
解析器	69	17	14	42	0.71	0.75	0.73
安全	129	0	8	5	1	0.38	0.55
设置	66	10	33	33	0.76	0.50	0.60
UI	4	15	0	123	0.89	1	0.94
总计	1200	200	157	715			

表15 混淆矩阵和性能：RTTS项目 - 迁移学习。

API领域	TN	FP	FN	TP	精确率	召回率	F值
APM	197	0	38	0	0	0	0
数据结构	100	0	135	0	0	0	0
DB	145	0	90	0	0	0	0
错误处理	223	0	12	0	0	0	0
事件处理	212	0	23	0	0	0	0
Lang	114	0	121	0	0	0	0
IO	44	21	131	39	0.65	0.22	0.33
i18n	214	0	21	0	0	0	0
日志记录	102	22	91	20	0.47	0.18	0.26
逻辑	13	17	146	59	0.77	0.28	0.41
微服务	28	0	206	1	1	0.004	0.009
网络	129	0	106	0	0	0	0
解析器	156	0	79	0	0	0	0
设置	115	14	98	8	0.36	0.07	0.12
线程	175	0	60	0	0	0	0
UI	3	38	26	168	0.81	0.86	0.84
总计	1970	112	1383	295			

日志记录 (3)。此外，被归类为不重要的标签包括：应用性能管理 (8)、设置 (7)、数据结构 (6) 和用户界面 (9)，以及安全 (2)。

在RTTS项目中，评估结果最好的四个标签（服务、日志记录、设置和网络）的精确率均高于0.75，而评估结果最差的四个标签中的两个（数据结构、用户界面、开发运维和语言）的精确率为 ≤ 0.7。一个阈值可以决定是否应报告某个标签。

Cronos 项目的参与者提到，他们希望看到已评估问题带有“数据结构”标签。这是因为在“数据结构”标签被用于 Cronos 项目中 90% 的入选问题后，我们将其移除了。一种解决办法是在项目描述中说明

表 16 混淆矩阵与性能：PowerToys 项目 - 迁移学习。

API领域	TN	FP	FN	TP	精确率	召回率	F值
APM	264	1	88	0	0	0	0
App	315	9	28	1	0.66	0.76	0.71
数据结构	344	3	6	0	0.03	0.40	0.06
i18n	209	134	4	6	0	0	0
解释器	342	1	10	0	0	0	0
日志记录	153	196	0	4	0.007	0.25	0.01
逻辑	173	27	100	53	0.01	1	0.02
微服务	0	348	0	5	0.25	0.49	0.33
解析器	6	105	10	232	0.07	0.13	0.09
设置	351	0	2	0	0.50	0.07	0.13
Test	174	82	56	41	0.006	0.66	0.01
UI	105	245	1	2	0.68	0.92	0.78

该项目高度依赖数据结构，因此报告的问题很可能涉及这方面的知识。

此外，参与者表示，一些标签可以为寻找缺陷的根本原因或确定解决新功能请求所需的工作提供线索。例如：“.....一些标签可以为所报告的问题提供一个初步的视角”(P2)或“网络：虽然对于这个特定案例而言，网络标签并不是必需的，但该问题可能是由服务之间的通信错误引起的，在这种情况下，它们本可以被.....”(P4)。另一方面，一些参与者不希望看到更通用的标签，例如数据结构或日志记录，因为它们出现在许多问题中：“数据结构无处不在，没有它们就不会有任何程序”(P1)，而另一些参与者则认为缺少数据结构标签（由于未达到90%的阈值，因此未出现在预测列表中），并建议将其包括在内（P14、P16和P17）。未来的工作可以确定如何处理开发者对于纳入通用标签的首选项。

本文所提出方法的泛化能力帮助我们涵盖了更多项目。然而，这也带来了一些问题。我们提出了能够适用于广泛项目类型的通用标签。这或许可以解释有关通用标签的评论。“.....这是一个与用户定义的配置数据相关的向后兼容性问题，因此从宽泛的解释来看，设置是准确的，但如果信息模型或领域模型存在，我会更倾向于使用它们。”（P1）。分析参与者对“验证”标签的建议时，我们回溯到这样一个阶段：自然语言处理相似度为与该问题相关的库提出了可能的 API领域，而专家也做出了选择。我们发现，所选的 API领域是“逻辑”，因为当时没有“验证”这一 API领域标签。如果专家来自该项目，那么或许可以存在“验证”这一 API领域标签，从而满足参与者的需求。

未来的工作可以探索更多 API 领域标签，以扩展并提出更多选项，从而适应其他项目。为项目定制标签可能会为所需技能提供更精确的方向，但也需要更多

专家工作时间。另一方面，泛化将该方法扩展到大量项目中，并可能降低 API 领域的含义层级。

对不同利益相关者有哪些实际影响? 新贡献者。 API 领域标签可以帮助开源贡献者，让他们能够提前了解处理问题所需的技能。这对新贡献者和休闲贡献者尤其有用 [72, 73],，他们此前没有使用该项目术语的经验。

项目维护者。 自动标注 API 领域可以帮助维护者根据所需的专业知识分配团队工作量，以处理项目任务。项目维护者还可以确定哪些类型的 API 会产生更多问题。我们的结果表明，我们可以预测最突出的 API 领域——在本例中为“实用工具”和“逻辑”——精确率分别高达 95% 和 91%（见表 13）。

平台/*Forge* 管理者。 参与者在选择要贡献的问题时，常会选择 TITLE、BODY和LABELS 来查找信息。我们的结果可用于改进问题列表页和详情页的布局，并使其优先级高于其他信息区域 (2)。例如，在 GitHub 的问题详情页中，标签信息显示在屏幕右侧，处于贡献者主要关注区域之外。

用于指导 GitHub 用户填写问题正文、以形成模式的模板，不仅有助于使不同问题之间的信息空间保持一致，也能帮助利用这些信息预测 API 标签的自动分类器。例如，我们研究中的一些错误预测，可能是由于标题和正文所含的有用信息太少，无法据此生成标签。

研究人员。 科学界可以将所提出的方法扩展到其他语言和项目，包括数据量更大、采用不同算法的项目。我们的方法还可用于改进相关工具，这些工具会根据新贡献者的技能和职业目标向其推荐匹配的任务（例如， [74]）。

教育工作者。 将开源软件贡献作为课程内容布置的教育工作者 [75] 也能从我们的方法中受益。对开源软件项目中的问题进行标注，有助于他们为课程选择示例或任务，从而为学习环境带来实践视角。

8 有效性威胁

有效性威胁分为“内部效度”“构念效度”和“外部效度”。

内部效度。 本研究有效性面临的威胁之一是 API 领域分类。我们承认，不同的个体可能会创建不同的分类，从而给结果引入一些偏差。为缓解这一问题，三位人员（包括两位资深开发人员和一位 JabRef项目的贡献者）创建了 API 领域标签类别，旨在使其适用于任何类型的项目。未来，我们可以通过协作方式（例如， [76, 77]）改进这一分类流程。

尽管不同背景的参与者参加了 JabRef 用户研究，但样本无法代表整个总体，结果可能存在偏差。研究随机为每位参与者分配了一个组。然而，部分参与者没有完成问卷，导致各组最终失去平衡。此外，我们创建子组的方式也可能给分析带来偏差。我们根据招募地点，将从业者分类为行业从业者或学生；但部分学生也可能是行业从业者，反之亦然。不过，这项分析的结果得到了按经验水平进行汇总的结果的佐证。

构念效度。另一个问题是我们数据集中的问题数量，以及问题与拉取请求之间的关联。为了将问题/键/-跟踪ID 纳入数据集，我们将其与通过拉取请求提交的解决方案（或“修订版”和“故障 ID”）关联起来。通过建立关联，我们能够识别用于创建标签的 API，并定义我们的真实标签（参见第 5.1.1节）。本研究无法识别在没有PR信息的情况下合并的问题。我们手动检查了随机抽取的问题样本（或“键”和“跟踪 ID”），以核实数据是否正确收集，以及是否反映了问题跟踪系统接口中显示的内容。两位作者从项目中随机抽取了 100 个任务进行手动检查，将收集的数据与 GitHub 接口进行比较。所有记录均保持一致，并且该验证集中的所有问题都已正确关联到其拉取请求。当关联数据存在多个对应项时，我们使用相应的语料库条目（标题、正文、评论、描述和摘要）连接所有数据。部分关联数据偶尔包含重复文本，可能导致我们的模型过拟合。未来版本可能会改进数据清理步骤。与其他项目不同，Cronos 通过以下列包含多个关联数据：“pai”和“ramo”、“关联问题”和“键”，以及“键”和“ramo”。这会造成一种递归情况，我们可能以不同方式将每次更新与多个“键”关联起来。我们倾向于保持简单，只使用与其他项目相似的关联数据：“键”和“ramo”。

在预测模型中，当预测模型呈现随机误差或噪声，而非潜在关系时，就会发生过拟合。在模型训练阶段，算法使用了测试集未包含的信息。为缓解这一问题，我们还使用了随机化方法来打乱训练样本和测试样本。

此外，我们承认并未调查这些标签是否帮助用户找到最合适的任务。用户研究并未评估 API 标签在帮助用户匹配自身技能方面的效果。我们的重点是了解 API 领域标签对参与者决策的相关性。此外，我们并未评估假阳性标签会如何影响任务的选择或排序。不过，我们认为这种影响很小，因为在 JabRef项目中推荐的 11 个问题里，参与者选择最多的三个问题中只有一个标签是假阳性。此外，当我们要求参与者选择带有 API 标签 + 项目标签（处理组）或项目标签（对照组）的问题时，可能会引入一些偏差。事实上，评估相关性差异时，每个.....

新（API领域）标签的出现所引入的偏差，可能会受到项目标签性能不佳的影响，从而掩盖测量实验中的差异。未来可以通过一项将贡献者与任务技能相匹配的实验来调查 API 标签的有效性，并识别错误分类造成的问题。让相关区域挑选问题的实证实验可能会引入偏差，因为参与者只是选择了一个问题，并没有解决该问题。

在对问题及其关联的拉取请求进行分类时，我们将变更的文件与项目最新版本中解析得到的源代码文件进行了比较。如果更新后的源文件已不存在，则舍弃该拉取请求。

外部效度。泛化也是本研究的一项局限。对于其他项目、编程语言生态系统，甚至使用不同语言编写的问题，研究结果都可能有所不同。为解决这一局限，我们沿着这一方向扩展了先前的研究 [14]，挖掘了不同的项目，涵盖三种编程语言和两种自然语言（或词汇体系）。尽管如此，本研究表明，多标签分类方法可用于预测 API 领域标签，也表明这类标签对新贡献者可能具有重要意义。此外，我们识别出的 API 领域标签也适用于其他使用相同 API、涵盖多个项目领域（桌面和 Web 应用程序）的项目。许多项目采用典型的架构（MVC）和框架（JavaFX、JUnit 等），因此与许多其他项目相似。正如 Qiu 等人所述 [70]，项目会采用常见的 API，其占所用 API 的比例最高可达 53%。此外，我们的数据可用作训练集，用于在其他项目中自动生成 API 领域标签。

9 结论

我们研究了新手是否会使用 API 领域标签来选择问题，以及新手会使用哪些信息来决定参与哪个问题。我们发现，行业从业者和有经验的程序员比学生和新手程序员更经常偏好使用 API领域标签。参与者更偏好 API 领域标签，而不是项目中已经使用的组件标签。用户希望看到包含问题类型、优先级、编程语言、复杂度、技术以及 API 信息的标签，并根据标题、正文、评论和标签来选择问题。

我们还研究了预测API 领域标签的程度。我们从五个项目的 22,231个问题中挖掘数据，并预测了 31 个 API 领域标签。分别对各项目进行训练和测试时，结合随机森林算法（RF）的 TF-IDF 和一元词获得了 84% 的精确率，超过了 BERT（精确率为 62%）。问题正文中的数据取得了最佳结果。然而，当共同预测所有项目的 API领域标签时，RF的精确率下降至 78%，而BERT的精确率提高至72%，这表明 BERT 技术应用于更大数据集时具有正向敏感性。

从多样化来源进行迁移学习导致评估指标下降，精确率和召回率的取值范围也较为广泛。未来的工作应研究如何确定何时或以何种方式在项目中应用迁移学习于 API 领域标签。

最后，开发者一致认为，最多 64.4% 的 API 领域标签对于识别技能十分重要，因此，如果这些标签可用，就应当有助于解决问题。

这项研究是朝着帮助新贡献者将其 API 技能与每项任务相匹配，并更好地确定适合开始其开源软件项目入门过程的任务迈出的一步。

致谢

本研究部分得到了美国国家科学基金会项目编号 1815486、1815503、1900903 和 1901031 以及 CNPq 项目编号 #313067/2020-1 的支持。我们还要感谢那些抽出时间回答问卷的开发者。

数据可用性

本研究期间生成和/或分析的数据集可在 Zenodo 存储库中获取[10]。

声明

利益冲突/竞争利益作者声明不存在利益冲突。

参考资料

1. J. Wang 和 A. Sarma，“我应该修复哪个缺陷：帮助新开发者加入新项目”，载于第 *4* 届软件工程协作与人文因素国际研讨会论文集。ACM，2011 年，第 76–79 页。2. I. Steinmacher、T. U. Conte 和 M. A. Gerosa，“理解并支持开源软件社区选择合适的起始任务”，载于*2015* 年第 *48* 届夏威夷国际系统科学会议。IEEE，2015 年，第 5299–5308 页。3. I. Steinmacher、M. A. G. Silva、M. A. Gerosa 和 D. F. Redmiles，“关于开源软件项目新手所面临障碍的系统性文献综述”，《信息与软件技术》，第 59 卷，第 67–85 页，2015 年。

[10] https://doi.org/10.5281/zenodo.6869246

4. I. Steinmacher、T. Conte、M. A. Gerosa 和 D. Redmiles，“新手在开源软件项目中提交首次贡献时面临的社会障碍”，载于第 *18* 届 *ACM* 计算机支持的协作工作与社会计算会议论文集，CSCW ’15 系列。美国纽约州纽约市：计算机协会，2015 年，第 1379–1392 页。5. C. Stanik、L. Montgomery、D. Martens、D. Fucci 和 W. Maalej，“一种基于 NLP 的简单方法，用于支持开源社区中的入门与留存”，载于*2018* 年 *IEEE* 软件维护与演化国际会议（*ICSME*）。IEEE，2018 年，第 172–182 页。6. T. Zimmermann、R. Premraj、N. Bettenburg、S. Just、A. Schroter 和 C. Weiss，“什么样的错误报告才是高质量的？”《*IEEE* 软件工程汇刊》，第 36 卷，第 5 期，第 618–643 页，2010 年。7. N. Bettenburg、S. Just、A. Schr¨oter、C. Weiß、R. Premraj 和 T. Zimmer- mann，“Eclipse 中错误报告的质量”，载于*2007* 年 *OOPSLA Eclipse* 技术交流研讨会论文集，eclipse ’07 系列。美国纽约州纽约市：ACM，2007 年，第 21–25 页。8. L. Vaz、I. Steinmacher 和 S. Marczak，“关于 Topcoder 软件众包中任务文档的实证研究”，载于*2019* 年 *ACM/IEEE* 第 *14* 届全球软件工程国际会议（*ICGSE*）。IEEE，2019 年，第 48–57 页。9. I. Santos、I. Wiese、I. Steinmacher、A. Sarma 和 M. A. Gerosa，“成败得失：新手识别 OSS 任务所需技能的能力”，载于*2022* 年 *IEEE* 软件分析、演化与再工程国际会议（*SANER*），2022 年，第 174–183 页。10. I. Steinmacher、C. Treude 和 M. A. Gerosa，“让我加入：帮助新手成功加入开源项目的指南”，《*IEEE* 软件》，第 36 卷，第 4 期，第 41–49 页，2018 年。11. F. Santos、B. Trinkenreich、J. F. Nicolati Pimentel、I. Wiese、I. Stein- macher、A. Sarma 和 M. Gerosa，“如何选择任务？新手与现有贡献者视角之间的不匹配”，《实证软件工程与度量》，2022 年。12. A. Barcomb、K. Stol、B. Fitzgerald 和 D. Riehle，“管理自由/自由开源软件社区中的偶发志愿者”，《*IEEE* 软件工程汇刊》，第 1–1 页，2020 年。13. “API 定义。”[在线]。可访问：https://languages.oup.com/ google-dictionary-en/ 14. F. Santos、I. Wiese、B. Trinkenreich、I. Steinmacher、A. Sarma 和 M. A. Gerosa，“我能解决它吗？识别完成 OSS 任务所需的 API”，载于*2021* 年 *IEEE/ACM* 第 *18* 届软件仓库挖掘国际会议（*MSR*）。IEEE，2021 年，第 346–257 页。15. J. Nam、S. J. Pan 和 S. Kim，“迁移缺陷学习”，载于*2013* 年第 *35* 届软件工程国际会议（*ICSE*）。IEEE，2013 年，第 382–391 页。16. M. Izadi、S. Ganji 和 A. Heydarnoori，“使用多标签分类算法为软件仓库推荐主题”，《实证软件工程》，第 26 卷，第 93 页，2021 年。

17. S. Vargas-Baldrich、M. Linares-V´asquez 和 D. Poshyvanyk，《使用字节码和依赖项自动标注软件项目》，载于*2015* 年第 *30* 届 *IEEE/ACM* 自动化软件工程国际会议（*ASE*）。IEEE，2015 年，第 289–294 页。18. X. Xia、D. Lo、X. Wang 和 B. Zhou，《软件信息网站中的标签推荐》，载于 *2013* 年第 *10* 届软件仓库挖掘工作会议（*MSR*）。IEEE，2013 年，第 287–296 页。19. B. Lin、F. Zampetti、G. Bavota、M. Di Penta 和 M. Lanza，《基于模式的问答网站意见挖掘》，载于*2019* 年第 *41* 届 *IEEE/ACM* 软件工程国际会议（*ICSE*）。IEEE，2019 年，第 548–559 页。20. G. Uddin 和 F. Khomh，《自动挖掘 Stack Overflow 上针对 API 表达的意见》，*IEEE* 软件工程汇刊，第 1–1 页，2019 年。21. G. Antoniol、K. Ayari、M. Di Penta、F. Khomh 和 Y.-G. Gu´eh´eneuc，《这是缺陷还是增强？一种基于文本的变更请求分类方法》，载于*2008* 年协作研究高级研究中心会议论文集：思想交流会，2008 年，第 304–318 页。22. N. Pingclasai、H. Hata 和 K.-i. Matsumoto，《使用主题建模将错误报告分类为缺陷和其他请求》，载于*2013* 年第 *20* 届亚太软件工程会议（*APSEC*），第 2 卷。IEEE，2013 年，第 13–18 页。23. Y. Zhou、Y. Tong、R. Gu 和 H. Gall，《结合文本挖掘和数据挖掘进行错误报告分类》，《软件：演化与过程》期刊，第 28 卷，第 3 期，第 150–176 页，2016 年。24. Y. Zhu、M. Pan、Y. Pei 和 T. Zhang，《是缺陷还是建议？一种自动标注问题的方法》，*arXiv* 预印本 *arXiv:1909.00934*，2019 年。25. F. El Zanaty、C. Rezk、S. Lijbrink、W. van Bergen、M. Cˆot´e 和 S. McIn- tosh，《自动恢复缺失的问题类型标签》，《*IEEE* 软件》，2020 年。26. Q. Perez、P.-A. Jean、C. Urtado 和 S. Vauttier，《是不是缺陷？这是个问题》，载于 *2021* 年第 *29* 届 *IEEE/ACM* 程序理解国际会议（*ICPC*）。IEEE，2021 年，第 47–58 页。27. R. Kallis、A. Di Sorbo、G. Canfora 和 S. Panichella，《工单标注器：由机器学习驱动的问题分类》，载于*2019* 年 *IEEE* 软件维护与演化国际会议（*ICSME*）。IEEE，2019 年，第 406–409 页。28. M. Izadi、K. Akbari 和 A. Heydarnoori，《预测软件仓库中问题报告的目标和优先级》，《实证软件工程》，第 27 卷，第 2 期，第 1–37 页，2022 年。29. J. Wang、X. Zhang 和 L. Chen，《预训练上下文语言表示在为 GitHub 问题推荐标签方面表现如何？》《知识型系统》，第 232 卷，文章编号 107476，2021 年。 [在线]。可访问：
https://
www.sciencedirect.com/science/article/pii/S0950705121007383 30. J. Devlin、M.-W. Chang、K. Lee 和 K. Toutanova，《BERT：用于语言理解的深度双向变换器预训练》，载于$NAACL$，2019 年。

31. Y. Park 和 C. Jensen，《超越精美图片：探究代码可视化对开源新手的益处》，载于第 *5* 届 *IEEE* 软件可视化与理解分析国际研讨会论文集，VISSOFT '09 系列。IEEE，2009 年 9 月，第 3–10 页。32. Y. Huang、J. Wang、S. Wang、Z. Liu、D. Wang 和 Q. Wang，《刻画并预测适合新手的优先问题》，载于第 *15* 届 *ACM/IEEE* 实证软件工程与测量国际研讨会论文集（*ESEM*），2021 年，第 1–12 页。33. M. Szumilas，《解释优势比》，《加拿大儿童与青少年精神病学学会期刊》，第 19 卷，第 3 期，第 227 页，2010 年。34. D. Sheskin，《参数与非参数统计程序手册》，第 5 版。Chapman & Hall，2020 年。35. T. Ohno，《丰田生产系统的诞生》，《日本经济研究》，第 10 卷，第 4 期，第 83–101 页，1982 年。36. H. Pacaiova，《使用 5W2H 方法分析并识别不合格产品》，质量中心，2015 年。37. A. C. T. Klock、I. Gasparini 和 M. S. Pimenta，《5W2H 框架：以用户为中心的游戏化设计、开发与评估指南》，载于第 *15* 届巴西计算机系统人因研讨会论文集，2016 年，第 1–10 页。38. S. Ducasse 和 D. Pollet，《软件架构重建：一种面向流程的分类体系》，*IEEE* 软件工程汇刊，第 35 卷，第 4 期，第 573–591 页，2009 年。39. A. Savidis 和 C. Savaki，《通过依赖图聚类和可视化从源代码中挖掘软件架构》，载于*IVAPP*，2021 年 12 月。40. "spacy 工业级自然语言处理。" https://spacy. io/，访问日期：2021-10-04。41. Y. Feng、J. Jones、Z. Chen 和 C. Fang，《使用多标签和问题转换技术对软件故障进行分类的实证研究》，载于*2018* 年第 *11* 届 *IEEE* 软件测试、验证与确认国际会议（*ICST*）。IEEE，2018 年，第 320–330 页。42. T. Guggulothu 和 S. A. Moiz，《使用多标签分类方法检测代码异味》，《软件质量期刊》，第 28 卷，第 3 期，第 1063–1086 页，2020 年。43. J. Ramos等，《使用 TF-IDF 确定文档查询中的词语相关性》，载于首届机器学习教学会议论文集，第 242 卷。新泽西州皮斯卡塔韦，2003 年，第 133–142 页。44. S. Ravichandiran，《*Google BERT* 入门：使用 *BERT* 构建和训练最先进的自然语言处理模型》。Packt Publishing Ltd，2021 年。45. D. Behl、S. Handa 和 A. Arora，《使用朴素贝叶斯和 TF-IDF 识别并分析安全缺陷的缺陷挖掘工具》，载于*2014* 年可靠性、优化与信息技术国际会议（*ICROIT*）。IEEE，2014 年，第 294–299 页。46. S. L. Vadlamani 和 O. Baysal，《研究 Stack Overflow 和 GitHub 中的软件开发者专业知识与贡献》，载于*2020* 年 *IEEE* 国际

全国软件维护与演化会议 *(ICSME)*。IEEE，2020 年，第 312–323 页。47. M. Izadi、A. Heydarnoori 和 G. Gousios，“使用多标签分类算法为软件仓库推荐主题”，实证软件工程，第26 卷，2021年09月。48. F. Herrera、F. Charte、A. J. Rivera 和 M. J. delJesus，多标签分类：问题分析、度量指标与技术，第1 版。Springer PublishingCompany, Incorporated，2016 年。49. A. Blanco、A. Casillas、A.P´erez 和A. D. de Ilarraza，“多标签临床文档分类：标签密度的影响”，专家系统与应用，第 138 卷，第112835 页，2019 年。50. F. Charte、A. J. Rivera、M. J.delJesus 和F. Herrera，“Mlsmote：通过合成实例生成处理不平衡多标签学习”，《知识型系统》，第 89 卷，第 385–397页，2015 年。51.M.-L. Zhang 和 Z.-H. Zhou，“Ml-knn：一种多标签学习的惰性学习方法”，模式识别，第40卷，第 7 期，第 2038–2048 页，2007年。52.“Fast BERT 代码仓库。”[在线]。可用地址：https://github.com/ utterworks/fast-bert 53.“Transformers 文档。”[在线]。可用地址：https://huggingface. co/docs/transformers/index 54.Y. You、J. Li、S. Reddi、J.Hseu、S.Kumar、S.Bhojanapalli、X.Song、J. Demmel、K. Keutzer和 C.-J. Hsieh，“深度学习的大批量优化：76 分钟内训练 BERT”，载于国际学习表征会议，2020 年。 [在线]。可用地址：https://openreview.net/forum?id=Syx4wnEtvH55. G. Tsoumakas、I. Katakis 和 I. Vlahavas，“挖掘多标签数据”，数据挖掘与知识发现手册，第 667–685 页，2009年。56. R. B. Pereira、A. Plastino、B. Zadrozny 和 L. H. Merschmann，“多标签分类性能度量的相关性分析”，信息处理与管理，第 54 卷，第 3 期，第 359–369 页，2018 年。57. M.Sokolova 和 G. Lapalme，“分类任务性能度量的系统分析”，信息处理与管理，第 45 卷，第 4 期，第 427–437 页，2009 年。58. J. Romano 、 J. Kromrey 、J.Coraggio 和 J. Skowronek，“序数等级数据的适当统计方法：我们真的应该使用 t检验和 Cohen's d 来评估 NSSE及其他调查中的组间差异吗？”载于佛罗里达州机构研究协会年度会议，2006 年，第1–3 页。59.C. Tantithamthavorn、S. McIntosh、A. E.Hassan 和 K. Matsumoto，“自动参数优化对缺陷预测模型的影响”，*IEEE* 软件工程汇刊，第 45 卷，第 7 期，第 683–711 页，2019年。60. D.Petkovic、M. Sosnick-P´erez、K.Okada、R. Todtenhoefer、S. Huang、N.Miglani 和 A. Vigil，“使用随机森林分类器评估和预测学生对软件工程团队合作的学习情况”，载于*2016 IEEE* 教育前沿会议 *(FIE)*。IEEE，2016 年，第 1–7 页。

61. E.Goel, E. Abhilasha, E. Goel 和 E.Abhilasha，“随机森林：综述”，《计算机科学与软件工程高级研究国际期刊》，第 7 卷，第1 期，2017 年。62. T.Pushphavathi、V.Suma 和 V.Ramaswamy，“软件缺陷预测的新方法：FCM 与随机森林的混合”，载于*2014* 年电子与通信系统国际会议（*ICECS*）。IEEE，2014年，第 1–5 页。63. S. M. Satapathy、B. P. Acharya 和 S.K.Rath，“基于用例点的随机森林技术进行软件工作量早期估算”，《*IET* 软件》，第 10 卷，第 1 期，第 10–17 页，2016 年。64. M. Van Gompel 和 A. VanDen Bosch，“使用 colibri core 进行高效的n-gram、skipgram 和 flexgram 建模”，《开放研究软件期刊》，第 4卷，第1 期，2016 年。65. T.Saito和M.Rehmsmeier，“在不平衡数据集上评估二分类器时，精确率-召回率图比ROC 图提供更多信息”，《*PloS one*》，第 10 卷，第 3 期，第 e0118432 页，2015年。66.P.A. Flach 和 M.Kull，“精确率-召回率增益曲线：正确进行 PR 分析。”载于*NIPS*，第 15 卷，2015 年。67. A. Strauss 和 J.Corbin，《定性研究技术基础》。美国加利福尼亚州千橡市：Sage出版社，1998 年。68. I. S.Wiese、R. R´e、I. Steinmacher、R. T. Kuroda、G. A. Oliva、C. Treude和 M. A. Gerosa，“使用上下文信息预测共同变更”，《系统与软件期刊》，第 128 卷，第220–235 页，2017 年。69. R. Agrawal、T. Imieli´nski 和A. Swami，“挖掘大型数据库中项目集之间的关联规则”，载于《*1993* 年*ACMSIGMOD* 数据管理国际会议论文集》，1993 年，第 207–216 页。70. D.Qiu、B. Li和 H. Leung，“理解Java 中的API 使用”，《信息与软件技术》，第73 卷，第81–100 页，2016 年。71. C.-W.Seah、I. W. Tsang 和 Y.-S. Ong，“迁移序数标签学习”，《*IEEE* 神经网络与学习系统汇刊》，第 24 卷，第11 期，第1863–1876 页，2013 年。72. G. Pinto、I.Steinmacher和 M. A.Gerosa，“比你想象的更常见：对临时贡献者的深入研究”，载于《*IEEE*第 *23* 届软件分析、演化与再工程国际会议，*SANER 2016*，日本大阪吹田，*2016* 年*3* 月 *14–18* 日——第*1* 卷》，2016 年，第 112–123页。73. S. Balali、I. Steinmacher、U. Annamalai、A. Sarma 和 M. A.Gerosa，“新手的障碍……就这些吗？对开源软件项目中导师和新手障碍的分析”，《计算机支持的协同工作》，第27 卷，第 3–6期，第 679–714页，2018 年12 月。74. A. Sarma、M. A. Gerosa、I. Steinmacher 和 R. Leano，“通过开源软件生态系统中的任务策划训练未来劳动力”，载于《*2016* 年第*24* 届 *ACMSIGSOFT* 软件工程基础国际研讨会论文集》，2016 年，第 932–935 页。

75. G. H. L. Pinto、F. Figueira Filho、I. Steinmacher 和 M. A. Gerosa，“使用开源软件训练软件工程师：教授的视角”，载于*2017 IEEE 第 30 届软件工程教育与训练会议（CSEE&T）*。IEEE，2017 年，第 117–121 页。76. M. Ferreira Moreno、W. H. Sousa Dos Santos、R. Costa Mesquita Santos 和 R. Fontoura De Gusmao Cerqueira，“通过 has 支持知识创建：超知识注释系统”，载于*2018 IEEE 国际多媒体研讨会（ISM）*，2018 年，第 239–246 页。77. Y. Lu、G. Li、Z. Zhao、L. Wen 和 Z. Jin，“学习从 API 文档推断 API 映射”，载于知识科学、工程与管理国际会议。Springer，2017 年，第 237–248 页。

A 附录

RQ2 结果的补充数据。第 5.2 节通过箱线图展示了部分数据。有关实验的冗余数据（以及更详细的数据）可在此处的表格中查阅。

表17 用于评估语料库的模型总体性能。Hla*

模型	语料库	精确率	召回率	F值	Hla
TD-IDF	标题 (T)	0.830	0.794	0.809	0.117
TD-IDF	正文 (B)	0.840	0.786	0.811	0.116
TD-IDF	T, B	0.839	0.799	0.817	0.113
TD-IDF	T, B, 评论	0.831	0.796	0.812	0.116
BERT	标题 (T)	0.616	0.592	0.596	0.277
BERT	正文 (B)	0.599	0.598	0.591	0.27
BERT	T, B	0.595	0.559	0.568	0.269
BERT	T, B, 评论	0.597	0.587	0.582	0.266

** Hla - Hamming Loss*

表 18 用于评估克数的模型的整体性能。Hla*

模型	精确率	召回率	F值	Hla*
一元词 (1,1)	0.841	0.829	0.834	0.115
二元语法 (2,2)	0.844	0.809	0.825	0.119
三元语法 (3,3)	0.841	0.809	0.822	0.123
四元语法 (4,4)	0.845	0.798	0.819	0.125

** Hla - Hamming Loss*

表19 用于评估算法的模型总体性能。Hla*

模型	Hla	精确率	召回率	F值
决策树	0.105	0.861	0.837	0.847
虚拟基线	0.202	0.749	0.658	0.698
逻辑回归	0.120	0.858	0.792	0.822
多层感知机分类器	0.107	0.853	0.846	0.848
多标签 k 近邻	0.126	0.837	0.801	0.816
随机森林	0.107	0.864	0.836	0.849
BERT	0.277	0.601	0.574	0.578

** Hla - Hamming Loss*

我们还给出了所有单独进行训练和测试的项目的混淆矩阵（表 21– 26）。RTTS 项目的混淆矩阵见第 7 节中的表 14。

表20 使用合并了所有项目的数据集创建的模型在评估算法时的总体性能

模型	Hla*	精确率	召回率	F值
决策树	0.157	0.768	0.576	0.654
逻辑回归	0.167	0.779	0.504	0.611
多层感知机分类器	0.154	0.766	0.595	0.666
多标签 k 近邻	0.179	0.709	0.555	0.617
随机森林	0.153	0.785	0.573	0.659
BERT	0.219	0.725	0.511	0.593

** Hla - Hamming Loss*

表21 所选模型的总体性能 - JabRef项目

API领域	TN	FP	FN	TP	精确率	召回率
网络	13	6	8	19	0.76	0.70
DB	43	1	0	2	0.67	1
解释器	12	9	5	20	0.69	0.8
日志记录	0	7	0	39	0.85	1
数据结构	45	0	0	1	1	1
i18n	40	0	5	1	1	0.17
设置	33	3	1	9	0.75	0.9
微服务	42	0	4	0	0	0
Test	41	0	3	2	1	0.4
IO	0	6	0	40	0.87	1
UI	4	2	0	40	0.95	1
App	41	1	2	2	0.67	1

表22 所选模型的总体性能——Powertoys 项目

API领域	TN	FP	FN	TP	精确率	召回率
APM	72	3	11	7	0.70	0.39
解释器	91	0	1	1	1	0.50
日志记录	92	1	0	0	0	0
数据结构	90	1	0	2	0.67	1
i18n	92	0	1	0	0	0
设置	47	6	11	29	0.83	0.72
逻辑	87	1	0	5	0.83	1
微服务	70	1	15	7	0.88	0.32
Test	91	1	0	1	0.50	1
搜索	41	6	8	38	0.86	0.83
UI	25	15	1	52	0.78	0.98
解析器	87	1	2	3	0.75	0.60
App	22	12	0	59	0.83	1

表23 所选模型的总体性能 - Audacity 项目

API领域	TN	FP	FN	TP	精确率	召回率
Util	38	3	2	13	0.81	0.87
APM	50	1	2	3	0.75	0.60
网络	54	0	0	2	1.00	1.00
DB	49	1	2	4	0.80	0.67
错误处理	37	3	2	14	0.82	0.88
日志记录	52	0	1	3	1.00	0.75
线程	46	1	0	9	0.90	1.00
Lang	54	0	0	2	1.00	1.00
数据结构	15	8	2	31	0.79	0.94
i18n	48	2	0	6	0.75	1.00
设置	13	5	2	36	0.88	0.95
逻辑	3	0	0	53	1.00	1.00
IO	10	3	6	37	0.93	0.86
UI	6	2	0	48	0.96	1.00
解析器	51	1	0	4	0.80	1.00
事件处理	28	6	1	21	0.78	0.95
App	29	4	4	19	0.83	0.83
GIS	50	1	2	3	0.75	0.60
多媒体	15	4	5	32	0.89	0.86
CG	50	0	1	5	1.00	0.83

表24 所选模型的总体性能 - MTT项目

API领域	TN	FP	FN	TP	精确率	召回率
NLP	34	0	9	9	1.00	0.50
APM	9	0	0	43	1.00	1.00
DB	37	0	4	11	1.00	0.73
Lang	10	1	2	39	0.97	0.95
开发运维	0	5	0	47	0.90	1.00
设置	18	6	8	20	0.77	0.71
IO	43	0	4	5	1.00	0.56
UI	0	2	0	50	0.96	1.00
安全	9	4	2	37	0.90	0.95

表25 混淆矩阵和性能：JabRef 项目 - 迁移学习。

API领域	TN	FP	FN	TP	精确率	召回率
网络	65	7	43	3	0.30	0.07
DB	107	8	3	0	0	0
解释器	71	0	47	0	0	0
日志记录	45	3	63	7	0.70	0.10
数据结构	98	16	4	0	0	0
i18n	103	0	15	0	0	0
设置	43	72	0	3	0.04	1
微服务	77	24	12	5	0.17	0.29
Test	77	13	23	5	0.28	0.18
IO	33	0	67	18	1.00	0.21
UI	2	35	13	68	0.66	0.84
App	9	90	0	19	0.17	1

表26 混淆矩阵和性能：项目 Audacity - 迁移学习。

API领域	TN	FP	FN	TP	精确率	召回率
APM	127	1	9	0	0	0
网络	132	1	4	0	0	0
DB	117	0	20	0	0	0
错误处理	105	0	32	0	0	0
日志记录	88	41	7	1	0.02	0.13
线程	122	0	15	0	0	0
Lang	133	0	4	0	0	0
数据结构	37	0	100	0	0	0
i18n	118	0	19	0	0	0
设置	28	1	94	14	0.93	0.13
逻辑	5	10	60	62	0.86	0.51
IO	31	8	65	33	0.80	0.34
UI	0	18	9	110	0.86	0.92
解析器	126	1	10	0	0	0
事件处理	68	0	69	0	0	0
App	58	33	12	34	0.51	0.74