一组通用 Python 代码变更模式的数据集

Akalanka Galappaththi加拿大埃德蒙顿阿尔伯塔大学
akalanka@ualberta.ca

Sarah Nadi阿尔伯塔大学，埃德蒙顿，加拿大nadi@ualberta.ca

arXiv:2304.04983v1 [cs.SE] 11 Apr 2023

摘要—从版本控制历史中挖掘重复的代码变更，是发现未知变更模式的一种常见方法。这类变更模式可用于代码推荐系统或自动程序修复技术。尽管已有针对Java 的此类工具和数据集，但在 Python 中寻找并推荐此类变更的工作还很少。在本文中，我们介绍了一个经过人工审查的、可泛化的 Python 重复代码变更模式数据集。我们制定了一份编码指南，用于识别可在自动化工具中使用的可泛化变更模式。我们利用近期从 Python 项目中挖掘重复变更的研究所挖掘出的变更模式，并使用我们的编码指南对这些模式进行人工审查。对于每项变更，我们还记录了变更描述及其应用原因，以及其他特征，例如该变更出现于多少个项目中。这一审查过程使我们能够识别并分享 72 个 Python 变更模式，这些模式可用于构建和推进 Python 开发者支持工具。

数据集链接：***https://figshare.com/articles/conference contribution/GeneralizablePython code change patterns/ 21966677***

I. 引言

软件系统会随着时间推移而不断演进，开发者会修改代码以实现新功能、修复错误，或寻找更简洁、易读或高效的方式来实现功能。开发者反复进行的源代码变更可能体现出有价值的改进，可作为代码推荐系统或自动程序修复工具的基础。例如，通过分析重复出现的变更，我们可以发现，在 Java 中，开发者经常用 addAll 方法 [1]替换逐个将给定集合中的元素添加到 list 中的循环。这种变更能使代码更加简洁易读，在某些情况下还能提高效率。

有大量研究利用这一思路，从版本控制历史中挖掘频繁出现的代码变更模式，并将这些模式用于推荐错误修复 [2]–[4], 、修复静态分析违规 [5], [6], 、增强代码 [4]–[7], 或迁移 API [8]。这类代码变更模式挖掘研究大多聚焦于 Java 项目 [1], [9]–[12] ，尽管 Python 是使用最广泛的编程语言 [14]，近期却只有一项研究关注 Python [13] 。

受上述 Java工具的启发，我们希望创建一种 Python 代码检查工具，用于标记频繁变更代码的实例，以提醒开发者有更好的实现方式。因此，我们研究了 Dilhara 等人 [13] 将 Java 频繁变更模式挖掘工具改编为适用于 Python 的近期工作。为了理解 Python

```
3 - sum = 0
4 - for element in elements:
5 -     sum += element
4 + sum = np.sum(elements)
```

图 1：一个将 for循环（更改前版本）替换为 API（更改后版本）的变更模式示例 [13]。

为了将代码演变作为开发者的学习资源，并创建一个可用于为代码变更建议奠定基础的数据集，同时构建自动化工具，作者分析了 1,000 个 Python 机器学习项目的历史记录，以挖掘代码变更模式。如果某个模式可以描述为高层次的程序转换，作者就认为该模式具有意义。图 1 展示了此类高层次程序转换的一个示例：用于计算元素之和的 for 循环被替换为对 numpy.sum 应用程序编程接口（API）函数的调用。通过分析，作者发现了 561 个有意义的代码变更模式 [15]。因此，我们着手将该数据集用作代码检查工具的基础。然而，在更仔细地考察这些变更模式时，我们发现其中一些模式并不代表可供自动化工具使用的通用变更。本文将变更模式定义为具有通用性，前提是变更后的代码与变更前的代码在语义上相似，且该变更具有普遍适用性。例如，检查图 1 中的源代码变更时，开发者可以发现，变更后版本第 4 行的 API 调用在语义上与变更前版本第 3 至 5 行相似。如果另一个项目具有类似的变更前代码结构（一个对迭代器中的项目求和的循环），就可以建议将其替换为相应的变更后版本。

图 2 展示了 Dilhara 等人 [13] 的 561 个代码变更模式中的两个示例，按照我们的定义，它们不具备泛化能力。在图 2a 的变更模式中，ax.plot(...) 和 plt.show() 是两个不同的操作。虽然挖掘算法可能找到了足够多的此类变更实例，从而将其标记为模式，但标记任何 ax.plot(...) 的用法并要求开发者将其替换为 plt.show() 是没有意义的，因为前者会创建图表，而后者会显示图表。换句话说，这些操作在语义上完全不同。图 2b 展示了另一个存在问题的变更模式。虽然该变更涉及相似的操作，只是在变更后的版本中使用了不同的矩阵类型，但此变更模式并不具有普遍适用性，

```
6  fig, ax = plt.subplot()
7  ax.fill(x, y)
8- ax.plot([x,[0], x[1]], [y[0], y[1]], linewidth = 2.0)
6  fig, ax = plt.subplot()
7  ax.fill(x, y)
8+ plt.show()
```

(a)将 plot API 替换为 show API (b)

```
7  A_indices = np.expand_dims(
8-   np.hstack((adj.row[:, None], adj.col[:, None])), 0
9    )
7  A_indices = np.expand_dims(
8+   np.hstack((adj.row[:, None], adj.col[:, None])).
       astype(np.int64), 0
9    )
```

图 2：无法泛化的变更模式 [13]

因为改用 np.int64 类型并非总是适用。其他开发者可能需要根据所处理的数据使用不同的类型，例如 int32 或 float32。

以上示例表明，虽然从项目提交历史中自动挖掘代码变更模式有助于发现此前未知的模式，但挖掘算法得出的模式并非总是可泛化或正确的。这个问题并非 Dilhara 等人 [13] 的研究所特有，而是频繁变更挖掘这一概念普遍存在的问题 [1],[9], [16]–[18]。这是因为挖掘算法会考察语法变更来发现模式，却不会考虑变更的语义。

因此，若要将变更模式用作任何自动化工具的基础，找到自动检测可泛化变更模式的方法就至关重要。于是，问题变成了：我们如何识别可用于提出建议的可泛化变更模式？在这篇数据论文中，我们采用一种定义明确且有文档记录的人工验证方法，从 Dilhara 等人 [13] 提供的数据集中识别可泛化模式。通过筛除不可泛化的模式，我们提供了一个数据集，其中包含可普遍应用于 Python 项目的代码转换。此外，我们还为这些模式添加了变更描述以及多种量化特征，例如应用某项变更的开发者人数、发现该模式的项目数量，以及对任何第三方 API 的调用情况。基于此，我们贡献了一个包含 72 个经过验证的 Python 变更模式的数据集，可用于提出代码变更建议，或推导用于识别可泛化代码变更模式的自动化标准 [19]。

II. 数据集构建

在本节中，我们简要概述工作中利用的现有数据集，以及我们为识别可泛化变更模式而开展的评估步骤。

A. 原始 *Python* 代码变更模式 *[13]*

Dilhara 等人 [13] 调整了 CPATMINER [1]，以发现 Python 项目中的代码变更模式。作者在来自不同领域的 1,000 个评分最高的 Python 机器学习应用项目的提交上运行该工具。自动挖掘过程得出了超过 28,000 个代码变更模式。每个模式至少包含三个变更实例，因为作者将频率阈值设为三，以便将重复的代码变更视为模式。为评估挖掘出的模式的实用性和性质，他们手动评估了一个选定样本：根据五个不同维度选取排名前 500 的模式，分别是频率、规模、频率 × 规模、发现该模式的项目数量，以及进行该变更的开发者数量。如果一个变更模式可以用高层次的程序转换来描述，作者就认为它是有意义的 。虽然没有提供明确的编码指南，但我们将“高层次的程序转换”理解为能够将变更描述为将 *X* 转换为 *Y*、将 *X* 替换为 *Y*或将 *X* 移入 *Y*。根据这一标准，作者从检查的 2,500 个模式中选出了 561 个模式。随后，他们将这 561 个变更模式归入不同的模式组，并将这些模式组归纳为趋势。表 I 的前三列概述了趋势、模式组以及各组中的模式数量。如引言所述，能够将代码变更描述为高层次的程序转换 [13] 还不足以排除无用的变更模式。

B. 识别有意义的变更模式

为了系统地手动调查可泛化的代码变更模式，我们考察了原作者已审核的 561 种代码变更模式。由于这 561 种变更模式是从五个不同维度中抽样选出的，因此相比从全部 28,000 种模式中随机抽样，它们构成了一个包含更可能稳定的代码变更模式的子集 [9], 。

为了减少我们人工调查的变更模式数量，我们首先筛除所有变更实例均由同一位开发者完成的模式。这是因为，如果多位开发者都认为有必要进行此项变更，那么该模式就更有可能适用且具有普遍性。根据这一标准，我们舍弃了 134 个模式，并继续分析剩余的 427 个模式。

现在我们继续手动审查剩余的 427 个变更模式。为了指导这一审查过程，我们需要制定一份定义明确的编码指南，并设定识别可泛化模式的清晰标准。为创建这份指南，我们随机选择了三个模式组（转换为 *list*或字典中的函数、替换数据可视化和更新容器），以识别模式的差异化特征。每个模式都包含一个示例代码片段，其中提供了变更前和变更后的版本，并重点展示模式挖掘算法所捕获的变更模式。图3 展示了一个消除用于更新 Python 字典值的 for 循环，并将其替换为Python内置 APIupdate 的模式。图3a 和图3b 中较深的文本表示变更模式中的变更，这些变更由挖掘算法捕获。

我们制定了以下编码指南，用于识别可泛化的代码变更模式。

1.更改前后版本的语义相似性：检查更改前后版本中发生更改的部分

```
def get_combined_dict(default_dict, additional_dict):
    d = default_dict.copy()
    for k, v in additional_dict.items():
        d[k] = v
    return d
```

(a) 使用for 循环更新字典的版本之前 (b) 使用更新

```
def get_combined_dict(default_dict, additional_dict):
    d = default_dict.copy()
    d.update(additional_dict)
    return d
```

图 3：使用 Python 内置 API 消解 for 循环的变更模式 [13]

代码反映语义相似的操作。如果否，则丢弃该模式。

2. 模式的普遍适用性：如果对标准 1 的回答是是，则检查该模式是否具有普遍适用性。如果该模式包含特定于项目的属性或函数，或者仅在特定情况下有效，则丢弃该模式。

我们使用编码指南中的标准 1 和标准 2，分别筛除图 2a 和图 2b 所示的代码变更模式。此外，如果模式的后续版本仅在特定情况下有效，编码指南中的标准 2 也会筛除这些模式。例如，如图 4 所示，我们不能总是建议开发者在转储*pickle* 文件时使用 tempfile.TemporaryDirectory 上下文管理器。总的来说，我们的编码指南确保标记为可泛化的代码变更模式能够作为代码变更建议提供给开发者。

为了使我们的数据集更适用于代码变更推荐系统，或用于识别可自动识别可泛化模式的标准，我们还会查找并记录所识别的可泛化变更模式的其他特征。我们会记录每种代码变更模式所使用的外部库和 Python 模块相关信息，以便筛选数据，为特定应用领域提供建议。例如，使用 PyTorch 或 TensorFlow 的可泛化代码变更模式可用于提供针对机器学习应用的代码变更建议。此外，我们还会记录每种代码变更模式涉及的开发者人数和项目数量。最后，我们会备注在标注过程中被筛除的模式。

在标注过程之后，我们研究标记为可泛化的模式，以了解为何进行该变更，以及为什么推荐此类变更可能有用。我们调查库/Python文档、在线问答论坛，以及属于每种模式的变更实例的提交消息，以收集有关该变更的信息。

III. 可泛化代码变更模式数据集

可泛化模式的数量： 我们发现，在 260 个（61%）模式中，变更后的代码不涉及语义相似的操作。在其余模式中，我们发现

```
3 - joblib.dump(pickle_file)
3 + with tempfile.TemporaryDirectory as tmp:
4 +     file_path = os.path.join(tmp, pickle_file)
5 +     joblib.dump(file_path)
```

图 4：使用上下文管理器和临时目录转储 *pickle* 文件的非可泛化模式示例 [13]。

95 个（22%）模式不适用于作为通用模式（根据编码指南中的标准 2 被剔除）。因此，我们得到 72 个（17%）可泛化的代码变更模式，可用于提出建议。

表 I 的第四列显示了各模式组中可泛化模式的数量和比例。我们发现，在 9 个模式组中，所包含的代码变更模式有 20%–100% 是可泛化的；其余 15 个模式组则没有可泛化模式，或其比例极低。

变更原因： 对于我们识别出的每种可泛化模式，我们都会附上自由文本描述和变更原因，并在表 I 的最后一列进行总结。模式组 1–8 涉及上下文管理器 [20]。模式组 1 涉及修改代码以使用上下文管理器，从而使开发者无需担心意外忘记关闭文件。同样，模式组 6–8 分别使用上下文管理器来包含依赖项、处理错误状态、处理引发的错误以及创建临时文件。

第二个趋势（模式组 9–15，11 除外）都侧重于运行时效率。与使用循环重写相同逻辑相比，使用 Python API 和 *NumPy* 等第三方库通常效率更高 [21]。例如，模式组 9 使用索引、均值和 einsum 等 API 来消除 for 循环，从而提升性能。同样，模式组 12 的可泛化模式使用 str.join 来拼接字符串，而不是在循环中逐个追加。

第三个趋势侧重于总体上更新 API 的用法，但我们发现，从模式组 16 中识别出的可泛化模式所涉及的更改能够提升性能（使用*Numpy.average* ，而不是通过用总和除以大小来计算均值）。相比之下，模式组 19 中的可泛化模式复合机器学习 *API* 使用新的 API 替换已弃用的 API。模式组 20 的两个模式之一更新容器使用列表容器来获取反转列表中的元素。模式组 20 的另一个可泛化模式不更改容器，而是将 dict 构造函数替换为 dict 字面量。我们认为，这种模式更适合归入将 *Dict*、*Set*、*List* 构造函数迁移为字面量。

第 22 组的可泛化变更模式通过使用 in 运算符来缩短条件语句。此模式着重于通过简化代码来提升可读性。第 23 组的所有可泛化模式都会将构造函数改为字面量，以改善运行时 [22]。此变更省去了额外查找 Python 内置函数（例如 list、字典、集合）的过程 [23]。第 24 组的模式中有许多可泛化变更模式使用 Python 推导式，也就是让代码更具 Python 风格的 Python 惯用法 [22]。这些模式还能提升代码可读性 [22]。

表 I：每个模式组中的可泛化模式数量。第1–3 列取自 Dilhara 等人 [13] ，而最后两列分别是我们通过审查过程确定的可泛化模式数量和示例原因。

#	趋势	模式组	# of 模式	可泛化对象的数量 模式（%）	示例原因
1	改用 with 语句并使用 上下文管理器	读取、写入和遍历数据	135	31（23%）	避免错误处理资源
2		禁用或启用梯度计算	27	0 (0%)	
3		切换机器学习训练设备	5	0 (0%)	
4		更改深度学习网络中的名称和变量作用域	30	0 (0%)	
5		执行 TensorFlow 计算图的依赖项	16	0 (0%)	避免错误处理依赖项
6		临时更改库的配置	7	1 (14%)	避免不当处理错误状态
7		迁移到 pytest 中的上下文管理器	85	6（7%）	避免不当处理抛出的异常
8		使用上下文管理器打开临时目录	28	2 (7%)	避免不当处理临时文件
9	溶解用于 将循环转换为领域 具体的 抽象概念	转换为 NumPy 中的优化操作	51	8（16%）	运行时效率
10		转换为*list* 或字典中的函数	7	2（29%）	运行时效率
11		转换为 Python 内置函数	5	0（0%）	
12		在*String*中转换为函数	4	1 (25%)	运行时效率
13		转换为集合 操作	2	1 (50%)	运行时效率
14		在 PyTorch 中使用优化操作	4	0（0%）	运行时效率
15		在 TensorFlow 中使用优化操作	4	1（25%）	运行时效率
16	更新 API	迁移到机器学习库中的 API	3	1（33%）	运行时效率
17		变换矩阵	13	0 (0%)	
18		切换数据可视化	4	0 (0%)	
19		复合机器学习 API	3	1（33%）	已弃用的 API
20		更新容器	15	2 (13%)	访问迭代器中的项
21		更新矩阵类型	4	0 (0%)	
22	使用高级 语言特性	简化条件语句	3	1 (33%)	提高可读性
23		将字典、集合、*list* 构造函数迁移为字面量	4	4 (100%)	运行时效率
24		转换为 Python*list*、字典、集合 推导式	101	10（10%）	以更具可读性、更符合 Python 风格的方式改进
总计			561	72 (17%)	

a) 库、开发者和项目：就库而言，我们发现，在已确认的可泛化模式中，有 9 个使用了外部库 *NumPy*、*TensorFlow*、*spaCy*、*h5py* 和*mpmath*。我们还发现，80% 的可泛化模式由 2名或 3名执行该变更的不同开发者完成。在考察发现泛化模式实例的项目数量时，这些模式中有 74%涉及 2个或 3 个不同项目。只有七个（9%）模式的实例提取自单个项目。

数据格式：我们在制品页面 [24] 上分享该数据集。我们提供了一个电子表格，其结构与表 I 类似。我们将每个可泛化模式归入相应的模式组。每种模式都包含一段说明变更原因的自由文本、一个指向更多信息的链接（例如该模式变更前后的版本），以及在原始挖掘过程中发现该模式的项目实例 [13]。此外，我们还提供了经过标注的原始数据集，其中也包含不可泛化的模式。

IV. 讨论与数据用途

本文介绍了一个经过人工验证的 Python 代码变更模式数据集。该数据集可用于提供 Python 代码变更建议、构建能够自动应用代码转换的工具，或探索可用于自动识别可泛化变更模式的差异特征。例如，将非惯用代码转换为惯用法的 Python 重构工具 [22] 可以利用该数据集中专属于 Python 惯用法的代码变更模式（例如模式组 24）来扩展其工作。

尽管与最初挖掘出的模式相比，我们提供的数据集规模较小 [13], ，但我们提供的 72 个模式均经过人工验证，确保具有普遍适用性。未经这一步审核，直接使用挖掘出的频繁变更模式可能会导致毫无意义或不正确的建议或转换，使开发者感到沮丧。因此，这项人工审查工作是构建实用工具的重要第一步。我们的数据集为其他研究人员节省了人工审查时间（35 小时），并为创建更多Python 开发者支持工具铺平了道路。最终的数据集包含多样化的模式，涵盖 15 个不同的模式组。数据集中的模式能够提高代码可读性、提升性能、提供更好的文件等资源处理方式，或介绍如何为已弃用的API 进行 API 迁移。我们记录的原因可以与代码变更建议一同提供，以增强开发者接受建议的信任度。

由于我们的数据既包含我们标记为可泛化的变更模式，也包含我们舍弃的变更模式，未来的研究可以探究如何推断它们之间的区别特征。寻找能够自动识别适用于自动化支持工具的变更模式的特征，有助于推动重复变更挖掘研究的发展 [4], [6], [18], [25]。

五、结论

我们审查了从 1,000 个 Python 项目的历史记录中挖掘出的 561 个现有变更模式 [13] ，并贡献了一个包含 72 个 Python 代码变更模式的数据集，同时提供了每项变更的描述和原因。总体而言，我们发现这些模式中只有 17% 足够通用，可用作代码推荐系统或自动化修复工具的基础。这凸显了在使用挖掘出的变更模式构建开发者支持工具之前，对其进行审查的必要性。因此，我们贡献的数据集 [24] 为构建更多自动化的 Python 支持工具铺平了道路。

致谢

我们谨此感谢 Malinda Dilhara 和 Danny Dig 对这项工作的反馈。

参考文献

[1] H. A. Nguyen、T. N. Nguyen、D. Dig、S. Nguyen、H. Tran 和 M. Hilton，“基于图的野外细粒度语义代码变更模式挖掘”，载于《第41届国际软件工程会议论文集》，ICSE 2019，蒙特利尔，魁北克省，加拿大，2019年5月25日至31日。IEEE / ACM，2019年，第819–830页。 [ ] 10 1109 2019 00089 在线。获取地址：https://doi.org/。/ICSE。J. Bader、A. Scott、M. Pradel 和 S. Chandra，“Getafix：学习自动修复错误”，《ACM程序设计语言论文集》，第3卷，OOPSLA期，第159:1–159:27页，2019年。 [在线]。获取地址：https: //doi.org/ 10 1145 3360585 。/ K. Pan、S. Kim 和 E. J. W. Jr.，“迈向理解错误修复模式”，《实证软件工程》，第14卷，第3期，第286–315页，2009年。 [ ] 10 1007 10664 008 9077 5 在线。获取地址：https://doi.org/。/s - - - N. Meng、M. Kim 和 K. S. McKinley，“LASE：通过示例学习定位并应用系统化编辑”，载于《第35届国际软件工程会议》，ICSE ’13，美国加利福尼亚州旧金山，2013年5月18日至26日。IEEE计算机学会，2013年，第502–511页。 [ ] 10 1109 2013 6606596 在线。获取地址：https://doi.org/。/ICSE。H. Yoshida、R. Bavishi、K. Hotta、Y. Nemoto、M. R. Prasad 和 S. Kikuchi，“Phoenix：一种针对静态分析违规自动合成数据驱动修复方案的工具”，载于《ICSE ’20：第42届国际软件工程会议，伴随卷》，韩国首尔，2020年6月27日至7月19日。ACM，2020年，第53–56页。 [在线]。获取地址：https://doi.org/ 10 1145 3377812 3382150 。/。Y. Ueda、T. Ishio 和 K. Matsumoto，“Devreplay：为重复代码变更生成正则表达式的代码检查工具”，《科学计算与编程》，第223卷，第102857页，2022年。 [在线]。获取地址：https://doi.org/ 10 1016 2022 102857 。/j.scico。O. Smirnov、A. Lobanov、Y. Golubev、E. Tikhomirova 和 T. Bryksin，“Revizor：一种基于图匹配自动处理频繁代码变更的数据驱动方法”，载于《第36届IEEE/ACM自动化软件工程国际会议》，ASE 2021，澳大利亚墨尔本，2021年11月15日至19日。IEEE，2021年，第1242–1246页。 [在线]。获取地址：https://doi.org/ 10 1109 51524 2021 9678635 。/ ASE。H. Alrubaye、M. W. Mkaouer 和 A. Ouni，“Migrationminer：一种在方法级别自动检测第三方Java库迁移的工具”，载于《2019年IEEE软件维护与演进国际会议》，ICSME 2019，美国俄亥俄州克利夫兰，2019年9月29日至10月4日。IEEE，2019年，第414–417页。 [在线]。获取地址：https://doi.org/ 10 1109 2019 00072 。/ICSME。H. A. Nguyen、A. T. Nguyen、T. T. Nguyen、T. N. Nguyen 和 H. Rajan，“软件演化中代码变更重复性的研究”，载于《2013年第28届IEEE/ACM自动化软件工程国际会议》，ASE 2013，美国加利福尼亚州硅谷，2013年11月11日至15日。IEEE，2013年，第180–190页。 [在线]。获取地址：https://doi.org/ 10 1109 2013 6693078 。/ASE。S. Negara、M. Codoban、D. Dig 和 R. E. Johnson，“挖掘细粒度代码变更以检测未知变更模式”，载于《第36届国际软件工程会议》，ICSE ’14，印度海得拉巴，2014年5月31日至6月7日。ACM，2014年，第803 813 [ ] 10 1145 2568225 2568317 –页。在线。获取地址：https://doi.org/。/。K. Herzig 和 A. Zeller，“纠缠代码变更的影响”，载于《第10届软件仓库挖掘工作会议论文集》，MSR ’13，美国加利福尼亚州旧金山，2013年5月18日至19日。IEEE计算机学会，2013年，第121–130页。 [在线]。获取地址：https://doi.org/ 10 1109 2013 6624018 。/MSR。B. Fluri、M. Wursch、M. Pinzger 和 H. C. Gall，“变更提炼：用于提取细粒度源代码变更的树差分方法”，《IEEE软件工程汇刊》，第33卷，第11期，第725–743页，2007年。 [在线]。获取地址：https://doi.org/ 10 1109 2007 70731 。/TSE。M. Dilhara、A. Ketkar、N. Sannidhi 和 D. Dig，“发现Python机器学习系统中的重复代码变更”，载于《第44届IEEE/ACM国际软件工程会议》，ICSE 2022，美国宾夕法尼亚州匹兹堡，2022年5月25日至27日。ACM，2022年，第736–748页。 [ ] 10 1145 3510003 3510225 在线。获取地址：https://doi.org/。/。TIOBE。一月的Tiobe指数。在线。获取地址：https: // www.tiobe.com/tiobe-index/ 2023 [ ]

[15]M. Dilhara、A. Ketkar 和 D. Dig。发现基于 Python 的机器学习系统中的重复性代码变更。 [在线]。可访问：https: //mlcodepatterns.github.io [16]K. Liu、A. Koyuncu、D. Kim 和 T. F. Bissyand´e，“AVATAR：使用静态分析违规的修复模式修复语义错误”，载于第*26*届 *IEEE* 软件分析、演化与再工程国际会议，*SANER 2019*，中国杭州，*2019*年*2*月*24*日至*27*日。IEEE，2019，第456–467页。 [在线]。可访问：https://doi.org/ 10 1109 2019 8667970 。/SANER。 R. Dyer、H. Rajan、H. A. Nguyen 和 T. N. Nguyen，“挖掘数十亿个抽象语法树节点以研究 Java 语言特性的实际和潜在使用情况”，载于第*36*届国际软件工程会议，*ICSE ’14*，印度海得拉巴，*2014*年*5*月*31*日至*6*月*7*日，P. Jalote、L. C. Briand 和 A. van der Hoek 编。ACM，2014，第779–790页。 [ ] 10 1145 2568225 2568295 在线。可访问：https://doi.org/。/。 A. T. Nguyen、M. Hilton、M. Codoban、H. A. Nguyen、L. Mast、E. Rademacher、T. N. Nguyen 和 D. Dig，“利用细粒度变更中的统计学习进行 API 代码推荐”，载于第*24*届 *ACM SIGSOFT* 软件工程基础国际研讨会论文集，*FSE 2016*，美国华盛顿州西雅图，*2016*年*11*月*13*日至*18*日。ACM，2016，第511–522页。 [在线]。可访问：https://doi.org/ 10 1145 2950290 2950333 。/。 A. Galappaththi 和 S. Nadi。通用 Python 代码变更模式。 [在线]。可访问：https://github.com/boneyag/msr23-artifact [20] P. 文档。with 语句上下文管理器。 [在线]。可访问：https://docs.python.org/3/reference/datamodel.html#context-managers [21] S. van der Walt、S. C. Colbert 和 G. Varoquaux，“NumPy 数组：一种高效数值计算的数据结构”，《计算机科学与工程》，第13卷，第2期，第22–30页，2011年。 [在线]。可访问：https://doi.org/ 10 1109 2011 37 。/MCSE。 Z. Zhang、Z. Xing、X. Xia、X. Xu 和 L. Zhu，“通过使用 Python 惯用法自动重构非惯用 Python 代码，使 Python 代码符合惯用法”，第696–708页，2022年。 [在线]。可访问：https://doi.org/ 10 1145 3540250 3549143 。/。 S. Overflow。Python 中的字面量与按名称实例化 list 和字典? [在线]。可访问：https://softwareengineering.stackexchange.com/questions/319547/字面量与按名称实例化-list-and-dicts-in-python [24] A. Galappaththi 和 S. Nadi。通用 Python 代码变更模式。 [在线]。可访问：https://figshare.com/articles/conference contribution/Generalizable Python code change patterns/ 21966677 [25] T. Zimmermann、A. Zeller、P. Weissgerber 和 S. Diehl，“挖掘版本历史以指导软件变更”，《*IEEE* 软件工程汇刊》，第31卷，第6期，第429–445页，2005年。