2024

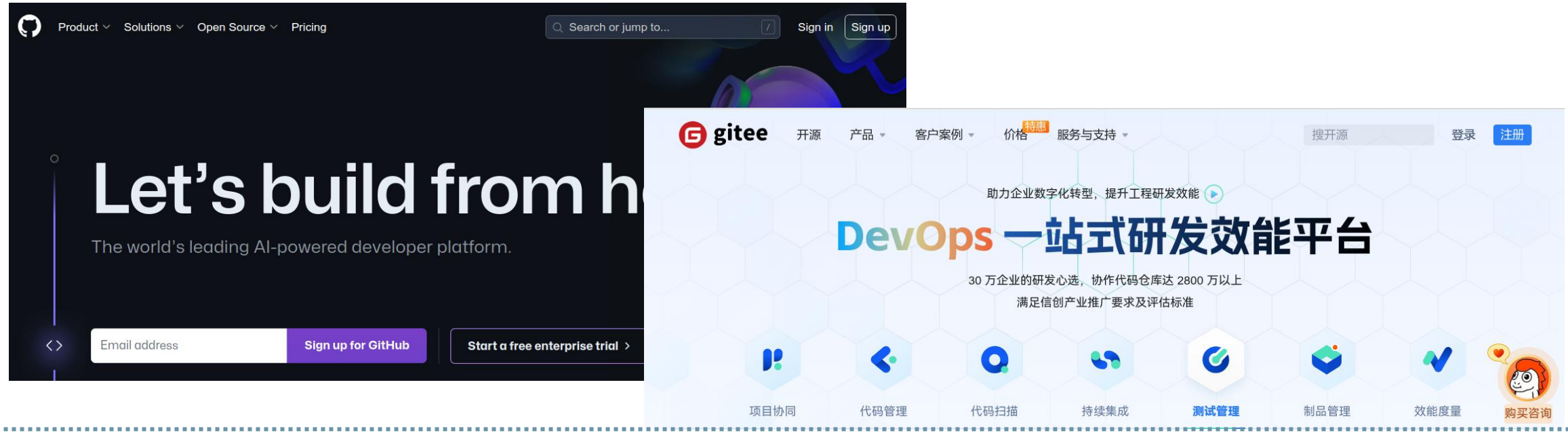
软件存储库的数据挖掘技术调研报告

1023040808-马张弛

目　　录

01 软件存储库挖掘简介

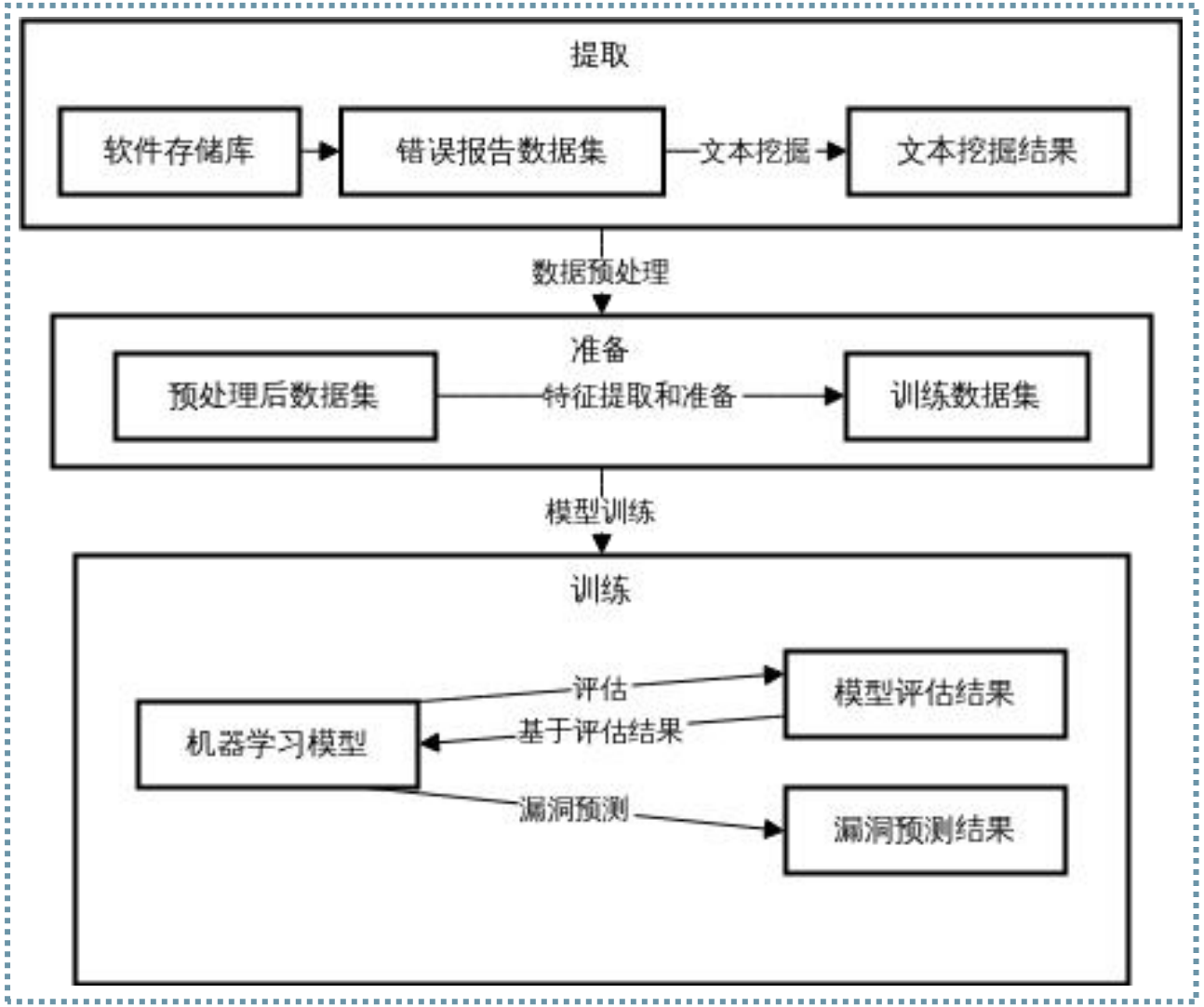
软件存储库



软件开发的过程中，会产生海量的数据，包括代码数据、错误报告以及代码的修改记录。得益于如GitHub、Gitee等协作开发的开源平台，这些开发过程中产生的数据的获取变得相对容易。通过对开发过程中产生的数据进行系统的整理和分析，可以从中提取出有价值的信息，进而为软件开发提供反馈，提升开发效率。

存储库挖掘

使用文本挖掘可以从错误报告中提取出关键信息，使用机器学习可以通过对历史数据的训练，建立模型来预测和识别潜在的漏洞，使用关联规则挖掘则可以从大量的数据中发现在代码修改和错误报告之间隐藏的关联性，使用聚类分析可以识别出相似的错误模式，从而更快地定位和修复问题。软件工程中的缺陷预测模型也可以基于历史数据构建，通过分析代码的复杂度、变更频率等特征，来预测哪些代码片段更有可能包含缺陷。



漏洞识别模型训练流程图

02

数据集构建相关工作

Minecraft: Automated Mining of Software Bug Fixes with Precise Code Context

	FixJs [3]	***Tufano et al.*** [4]	***Minecraft*** (this paper)
Introduced in	MSR 2022	TOSEM 2019	This paper
Language	JavaScript	Java	C, C++, Python, and Java
Granularity	Function	Function	File, Function, and Statement (ranges)
#Commits	~2M	~787K	~2.2M
#Bug-Fixes	300K	~2.3M	~3.29M
Dataset size	5.47GB	~7GB	28.8GB
Bug Detection Strategy	Commit messages with 6 keywords [5]: [“fix”, “solve”, “bug”, “issue”, “problem”, “error”]	Commit messages with 6 keywords [5]: [“fix”, “solve”, “bug” “issue”, “problem”, “error”]	Commit messages with 52 keywords [6] (*c.f.* Sec. III-C) + RegEx. [7]: `‘.((solv(ed|es|e|ing))|(fix(s|es|ing|ed)?)’|((error|bug|issue)(s)?)).’` + SZZ Algorithm [8]

- 从数据集的角度来看，虽然能获取海量开发过程数据，但是为了未来的使用与训练，必然需要特征提取、数据清洗等数据预处理环节。好的数据集可以为算法提供更好的训练效果、更准确的判断结果、更合理的评估指标
- 现有的数据集在涵盖的编程语言和包含的项目数量方面受到限制，并且这些数据集缺乏精确的错误位置和有关错误类型的信息（粒度不够细）。获取错误修复提交所使用的关键字过少
- 针对现有数据集的问题，作者通过提供一种解决方案来自动、精确地从开源存储库中自动、精确地挖掘包含错误、修复、精确位置和错误类型的代码片段，从而解决了现有的限制。提出的MINECRAFT数据集包含精确的错误位置和上下文，甚至可以找到方法范围之外的错误

工作流程

第一步：项目筛选

stars数量最低门槛为 1000 stars,语言包括Python 、C/C++。规定提交次数在1206到25000的范围。

第二步：提取已修复错误的提交

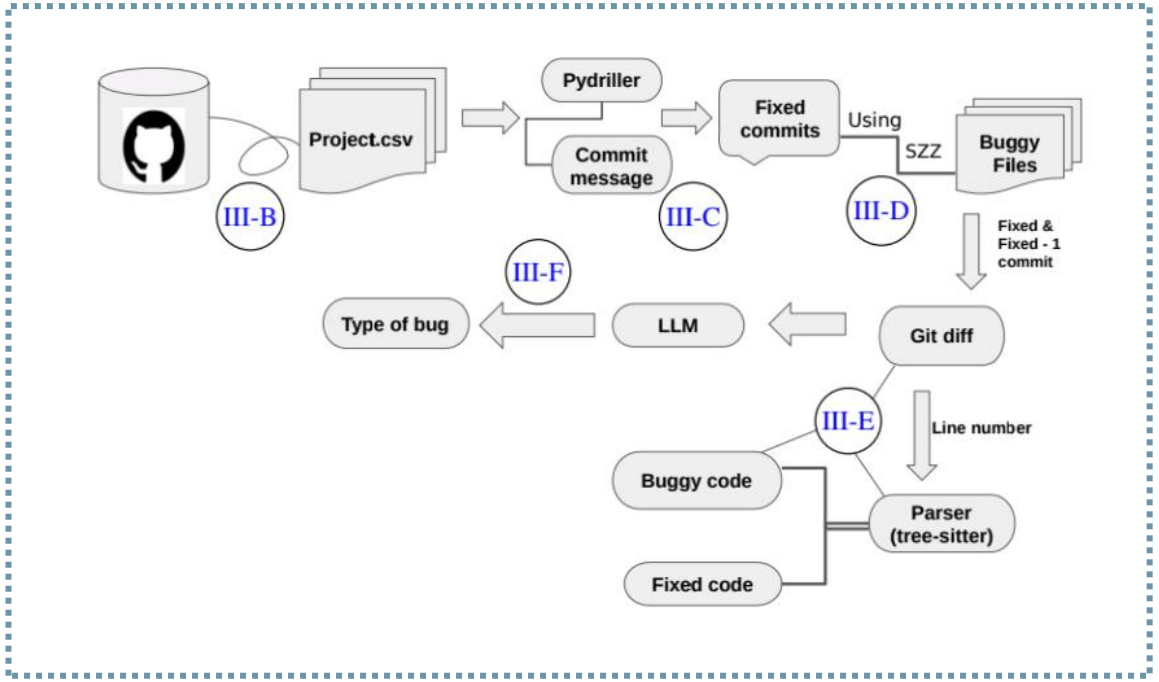
作者使用 PyDriller 从每个项目中提取提交，然后使用提交消息的关键词如fixed、bug，再使用正则表达式来识别错误修复提交。

((solv(ed|es|e|ing))|(fix(s|es|ing|ed)?)|((error|bug|issue)(s)?))

第三步：使用SZZ 算法和 git diff

PDriller提供的SZZ算法寻找错误修复的位置。

通过比较fixed提交与fixed-1提交之间的差异来识别导致引入错误的特定代码更改。



第四步：确定位置、错误代码和修复代码。

对git diff得到的结果进行过滤。使用tree-sitter解析器从源代码中检索相应的行，从而隔离 bug 及其后续修复的代码部分。

第五步：识别bug类型

作者使用Commit Message Generator 大语言模型来对错误近似分类。

总结

作者的方法首先数据量大，其次兼容的语言足够多，且使用的错误识别方法粒度更细，可以识别到更多的错误。作为工具，Minecraft数据集可以用于自动修复程序的数据集，一方面是参考数据集，一方面可以用于训练模型。作为思路，Minecraft数据集所使用的数据清洗组织方法可以借鉴，用于创建自己的错误修复数据集。

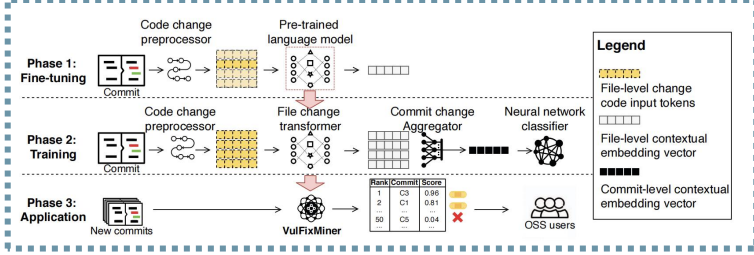
局限性在于，作者的研究结果可能不会超出所研究项目和特定编程语言的范围。据集的错误验证依赖于所使用的数据收集工具的准确性，并且错误修复提交中的人类标签的固有假设是真实的，因此需要手动验证来减少上述威胁。尽管经过严格的测试和验证，作者的实现脚本可能并不完全没有错误或错误。对于实际使用来说，首先数据集非常庞大，所需的存储和计算条件都比较严格，需要比较好的硬件条件。其次数据搜寻方法的漏洞会在使用过程中暴露出来，使用数据集训练的模型也会存在相关问题，需要未来解决。

03 自动识别漏洞修复相关工作

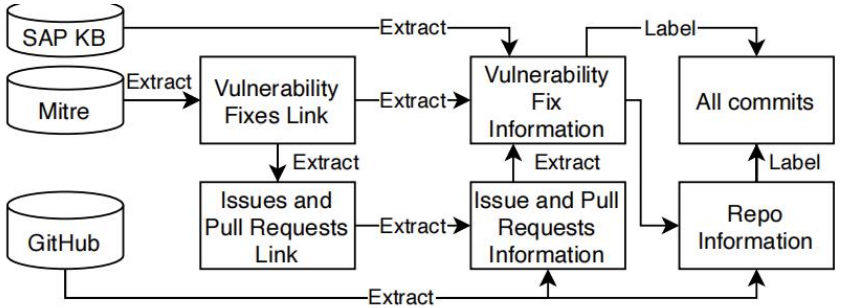
Finding A Needle in a Haystack: Automated Mining of Silent Vulnerability Fixes

- 开源软件由于更新频率高、学习成本低、使用成本低等原因，得到了广泛使用。开源软件在漏洞被披露之前不公开修复程序，会延长漏洞的暴露时间，使OSS用户无法防御安全攻击。2017 年，由于错过安全更新，Equifax 遭遇数据泄露，超过 1.43 亿美国消费者的个人信息遭到泄露。尽早了解修复程序有助于缩短 OSS 用户的漏洞修复时间。
- 已有的研究利用提交消息和问题报告来识别漏洞修复。然而漏洞修复主要体现在代码修复中，只使用提交消息和问题报告准确率不够高。与之前的工作不同，作者提出了 VulFixMiner，一种基于 Transformer 的模型，用于在实际环境中自动识别静默漏洞修复。VulFixMiner 能够跨项目、跨语言及时识别漏洞。

工作流程



第一步：数据收集



第二步：预训练

在预训练阶段，选择CodeBERT作为预训练语言模型，提取文件内的代码变更，然后将代码行标记为令牌序列，使用三个分隔符令牌进行连接。

第三步：微调与训练

接下来进行微调以最小化交叉熵损失，微调后的CodeBERT可以将输入的文件转为上下文嵌入向量。

在训练阶段，文件级输入令牌聚合为对应文件的向量，计算向量平均值，输入到全连接层，通过输出层进行分类。

	Training Set			Validation Set			Testing Set		
	#V.F.	#N.V.F	#Projects	#V.F.	#N.V.F	#Projects	#V.F.	#N.V.F	#Projects
Java	983	31,323	120	191	6,921	119	300	87,856	30
Python	522	20,362	84	80	2,949	83	195	55,638	22

第四步：应用

在应用阶段，VulFixMiner可以集成到OSS代码仓库的自动监控服务中。给定一组新的提交，VulFixMiner为每个提交计算分数，并为OSS用户提供一个提交排名。提交按分数排名，一个提交的分数越高，表明该提交用于修复漏洞的概率越高

评估与总结

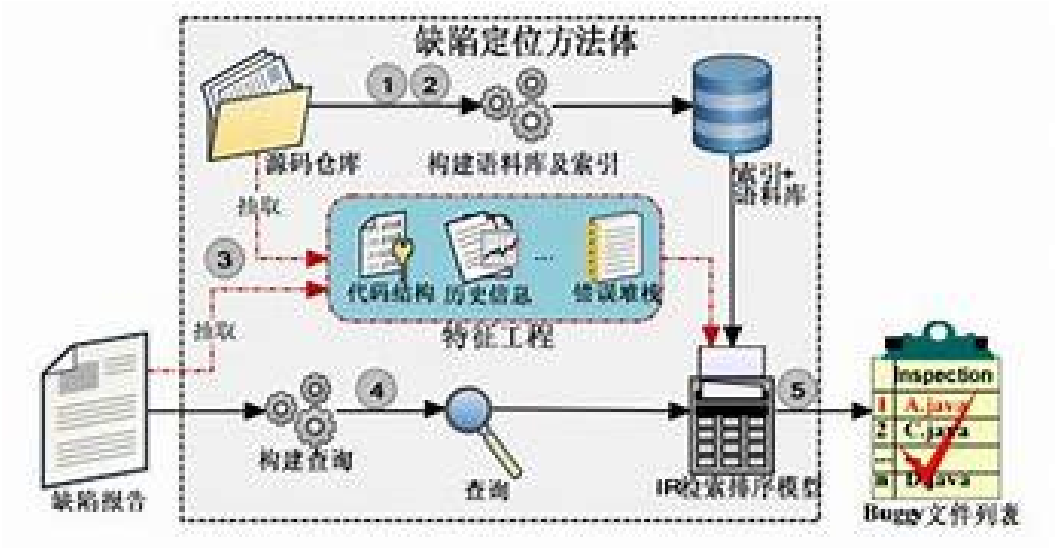
比较之后，VulFixMiner在所有评估性能指标上均优于Java和Python项目中的所有基准模型。

对于Python项目，VulFixMiner实现了AUC、CostEffort@5%、CostEffort@20%、Popt@5%和Popt@20%分别为0.73、0.32、0.56、0.24和0.39。与所有基准模型进行比较，这些结果分别提高了11-23%、18-26%、18-41%、16-21%和18-30%。对于Python项目，VulFixMiner实现了AUC、CostEffort@5%、CostEffort@20%、Popt@5%和Popt@20%分别为0.73、0.32、0.56、0.24和0.39。与所有基准模型进行比较，这些结果分别提高了11-23%、18-26%、18-41%、16-21%和18-30%。

这个方法可以帮助用户更好地维护开源软件，更放心地使用它。基于机器学习的方法也属于新领域，有更多的研究空间。但是训练模型需要电脑性能支持，其次数据集的收集和整理比较困难。机器学习的模型在适配性上较弱，个性化并投入使用存在一定难度。

04 错误定位技术相关工作

IncBL: Incremental Bug Localization



- 基于信息检索的错误定位（IRBL）是软件工程中的一个热门研究课题，并在过去十年中取得了可喜的成果。随着软件存储库的不断发展，即使是一些微小的更改，这些工具都需要重新处理所有文件并更新整个模型表示以确保准确性，这限制了它们在时间敏感或计算密集型场景中的使用。并且当前没有可供从业者采用的开源工具。

- 2004 年至 2019 年间发布的 IRBL 工具分为三代。仅仅基于 BoW 模型的第一代工具表现最差。第二代工具使用代码库中的结构信息和软件演化信息（例如历史错误报告）来提高准确性例如BugLocator 和BLUiR。 第三代工具利用术语顺序和语义，例如SCOR，需要大量时间和计算资源来训练模型。作者为了提高 IRBL 的效率，且保持良好的准确性，并解决业内没有相关可用软件的问题，开发一种增量错误定位工具

工作流程

第一步：处理代码文件

BugLocator利用 Java 解析器提取标识符，附加到原始代码内容中。然后，BugLocator 执行词干提取和停用词删除以生成代码语料库。最后，使用向量空间模型 (VSM)对代码语料库中的每个文档进行向量化，以便可以执行进一步的步骤。

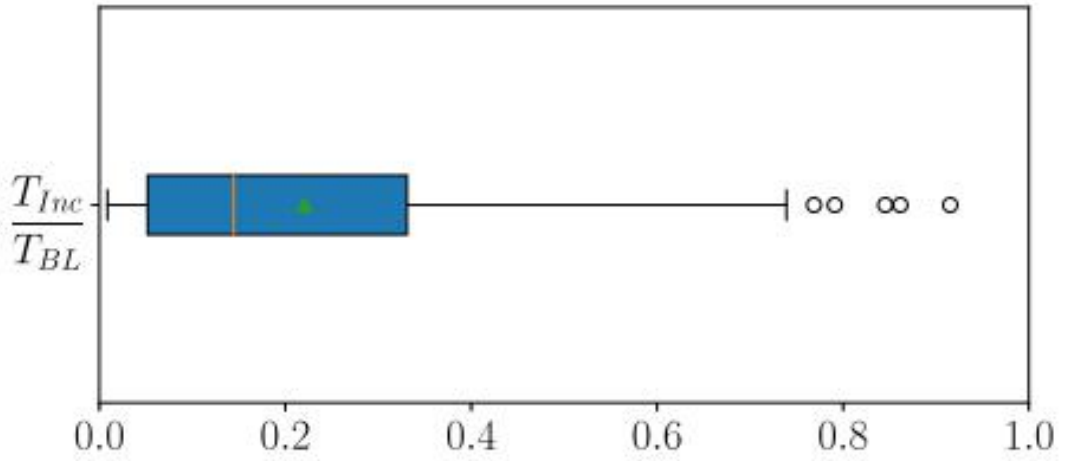
第二步：处理错误报告

首先将错误报告标题与描述结合起来。然后，对组合文档执行词干提取和停用词删除后，创建错误语料库。BugLocator通过使用VSM计算错误报告与所有过去修复的错误报告之间的相似性，利用错误修复历史信息来帮助对错误文件进行排名。当前的新错误报告和过去类似的已修复错误报告很可能共享相同的要修改的源代码文件。

第三步：本地化存在bug的文件

定义并计算错误报告和源代码文件之间的相关性分数。丢弃出现在错误报告中但未出现在代码语料库中的所有术语，然后使用步骤 1 中生成的 VSM 来计算源代码d与bug报告的 VSMScore。此外，使用函数 g 对 VSMScore 进行加权来修改 VSM 模型，以在排名期间支持长文档。

评估与总结



作者测量了IncBL定位潜在错误文件所需的运行时间与BugLocator每个错误报告的运行时间。作者从所有研究的错误报告中随机抽取了381份报告，考虑到95%的置信水平和5%的区间，形成具有统计代表性的样本。作者在这381个样本报告上测量了IncBL和BugLocator的运行情况。在测量准确性时，作者保持了与Bugzbook实验相同的设置：将一组错误报告与一个软件版本相关联。作者对所有43,017个错误报告运行了IncBL，并增量更新了每个新软件版本的VSM表示。IncBL 对所有研究的 bug 报告执行 bug 定位的运行时间与原始 BugLocator （ TInc ） 的运行时间比率的箱线图中，中位数和平均值分别为 14.44% 和 22.21%，对应于 6.30 和 4.50 倍的加速。与原始 BugLocator 相比，在保持相同精度的情况下，IncBL 可以将在不断发展的存储库中定位错误的运行时间平均减少 77.79%（即快 4.5 倍）。

THANK YOU