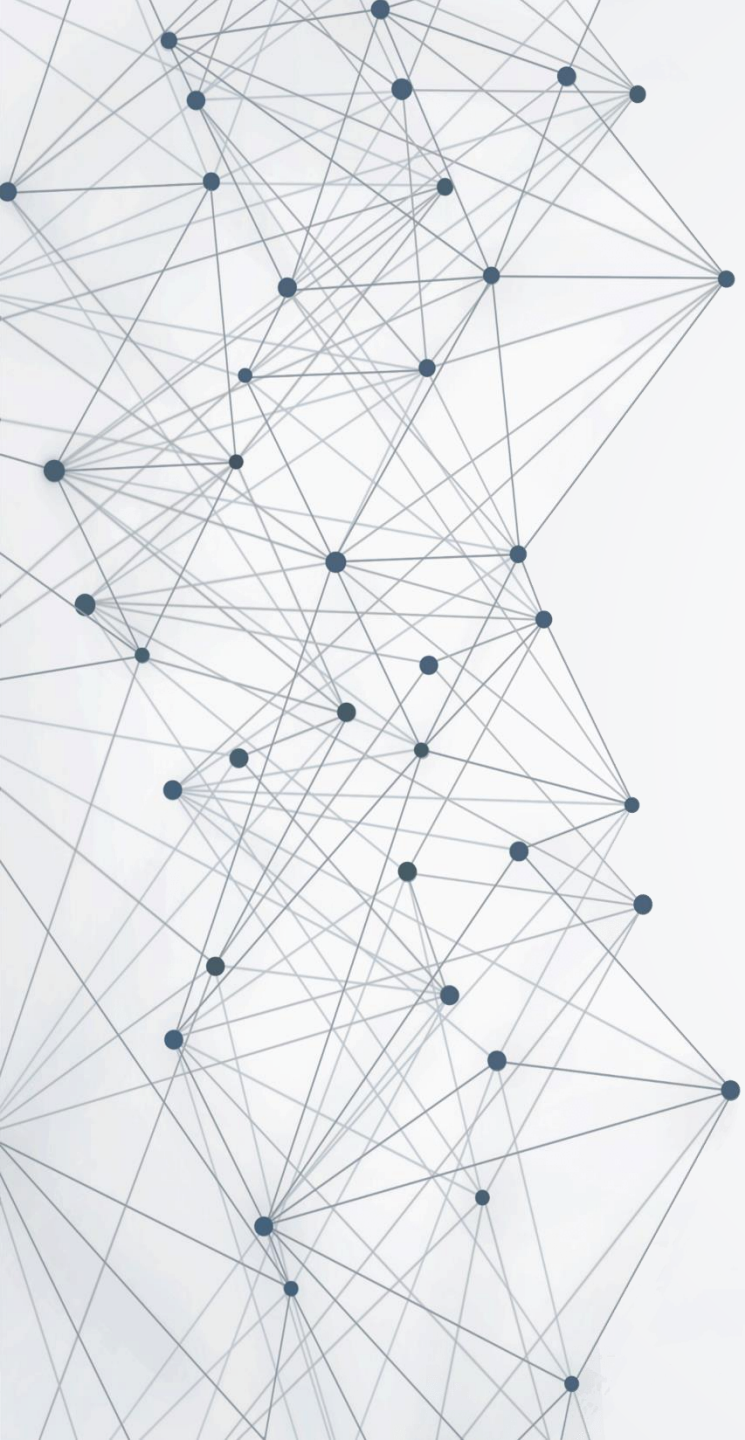


自动化注释与代码一致性检测方法

汇报人： 欧阳卓文

学号：1023040926

CONTENTS

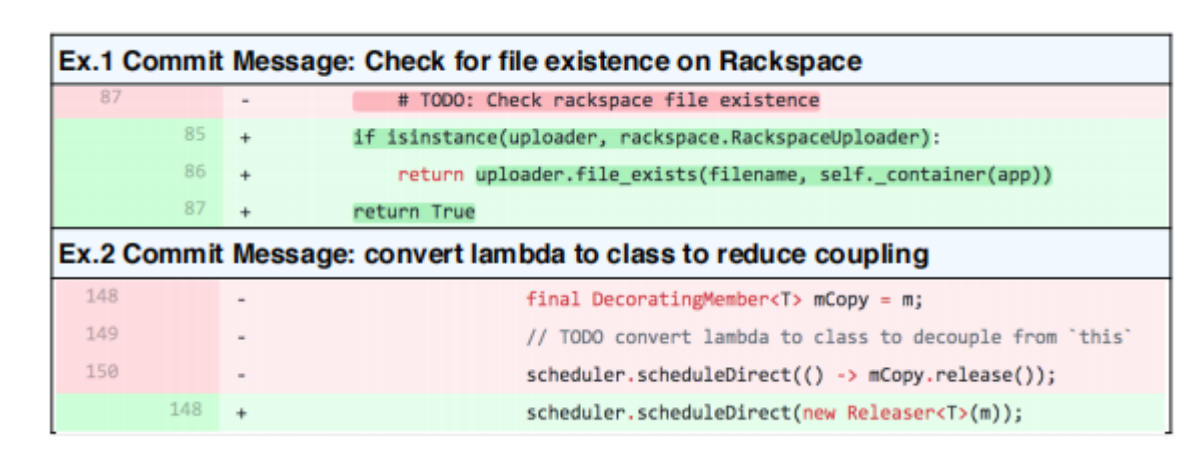
Part one

识别过时TODO注释

Trust me

研　究　问　题

TODO注释是开发人员在软件开发过程中广泛使用的一种方式，用于描述他们待完成的任务。然而，在完成任务后，开发人员有时会忽略或遗忘删除TODO注释，导致注释过时。过时的TODO注释可能会混淆开发团队，并可能在未来引入错误，降低软件的质量和可维护性。手动识别过时的TODO注释费时费力，因此有必要在它们引发任何不必要的副作用之前自动检测并删除它们。



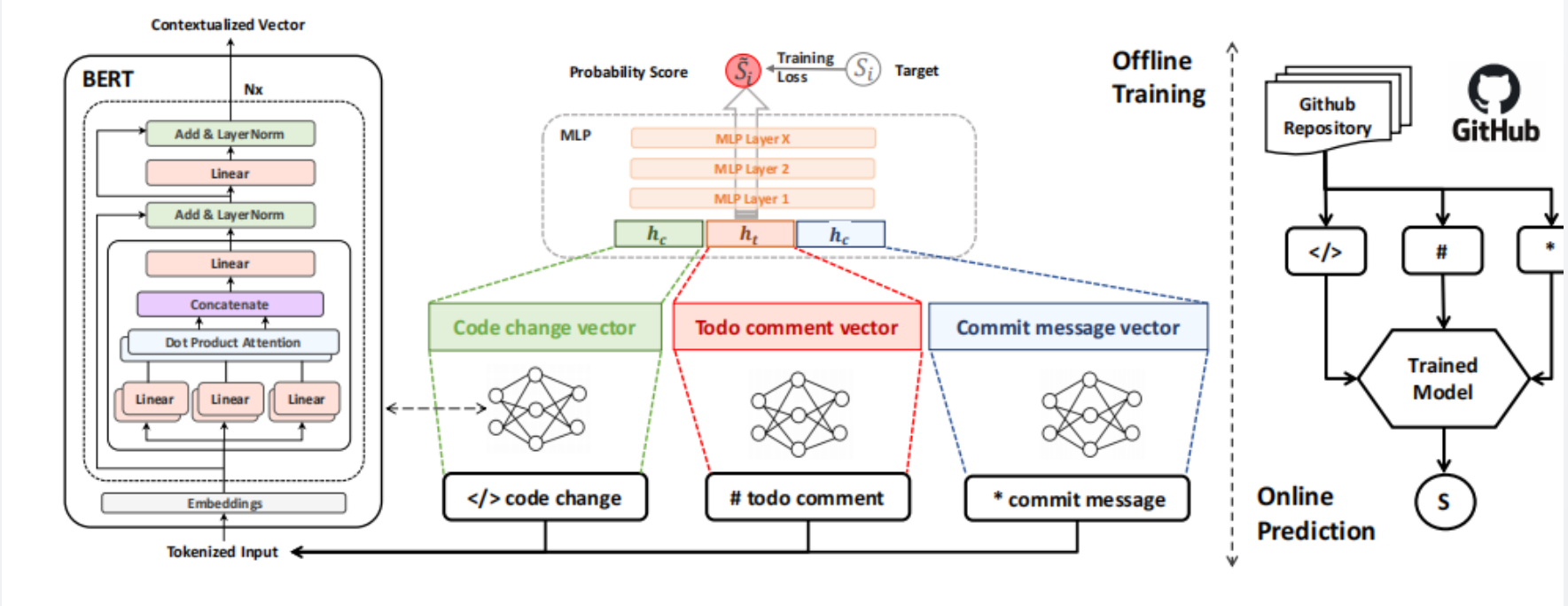
《Automating the Removal of Obsolete TODO Comments》提出了一种名为TDCleaner的模型用于识别软件项目中的过时TODO注释。TDCleaner能够帮助开发人员及时检查TODO注释的状态，并避免留下过时的TODO注释。论文在收集了来自前10000个Python和Java Github仓库的TODO注释的数据集上对TDCleaner进行了评估，并与一组基准进行了比较。广泛的实验结果显示了该模型在多个基准上的良好性能。论文还对真实世界的软件项目进行了实地评估，并向Github开发人员报告了18个经TDCleaner识别出的过时TODO注释，其中9个已经被确认并删除，证明了该方法的实际有效性。

相　关　方　法

本文提出了一种名为TDCleaner（TODO注释清除器）的新方法，用于自动识别软件项目中过时的TODO注释。TDCleaner的主要目标是帮助开发人员及时检查TODO注释的状态，并在它们引起任何不必要的副作用之前自动删除它们。该方法包括离线学习和在线预测两个主要阶段。

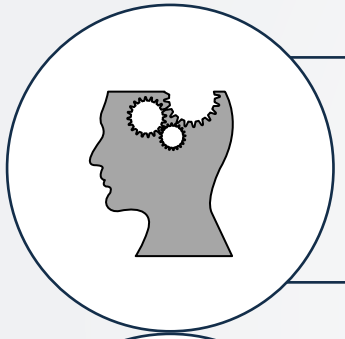
在离线学习阶段，首先自动建立了训练样本，包括代码更改、TODO注释和提交消息之间的关系。为了捕捉TODO注释、代码更改和提交消息的语义特征，使用了三个神经编码器分别对它们进行编码。TDCleaner通过优化最终的概率分数来自动学习不同编码器之间的相关性和交互作用，从而估计TODO注释的最终状态。

在在线预测阶段，通过利用离线训练好的模型判断TODO注释是否过时。对于给定的TODO注释，将其与相关的代码更改和提交消息配对，并将它们输入训练好的TDCleaner模型，估计它们的匹配分数，从而判断TODO注释的状态。



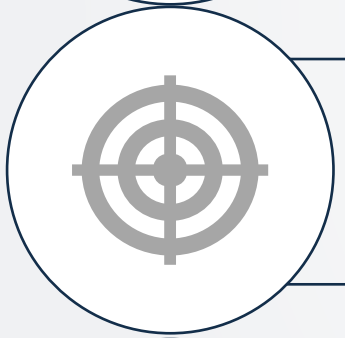
小 结

作者从Python和Java的Github仓库中收集了TODO注释数据集，并对TDCleaner进行了广泛的实验评估，结果显示了该模型在多个基准测试上的优越性能。但作者也提到了一些局限性。



无法处理多个TODO注释

当前的方法在分析未见数据时缺乏处理多个TODO注释的能力。



仅基于Python和Java项目

作者构建的数据集仅来自GitHub上的Python和Java项目。这限制了该方法在其他编程语言项目上的适用性。

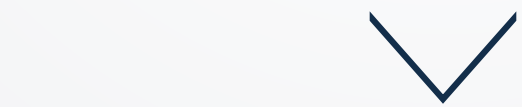
仅关注TODO注释

该方法仅关注TODO注释，而未考虑其他类型的自我承认的技术债务（SATD），如HACK、FIXME等。这可能限制了方法的适用性和泛化能力。

Part two

自动化代码注释更新

Trust me



在软件开发中，代码注释对于程序理解和软件维护非常重要，并且随着代码的演化也需要进行相应的维护。然而，当修改代码时，开发人员有时会忽视更新相关的注释，导致注释与代码不一致或过时（即糟糕的注释）。这样的注释会误导开发人员，并可能导致未来的错误。因此，修复和避免糟糕的注释是必要的。

```
public Map<MetricName, ? extends Metric> metrics() {
    ...
    for (final StreamThread thread : threads) {
        result.putAll(thread.producerMetrics());
        result.putAll(thread.consumerMetrics());
        result.putAll(thread.adminClientMetrics());
    }
    ...}
```

Method Comment: Get read-only handle on global metrics registry, including streams client's own metrics plus its embedded consumer clients' metrics.

Updated Comment: Get read-only handle on global metrics registry, including streams client's own metrics plus its embedded producer, consumer and admin clients' metrics.

《Automating Just-In-Time Comment Updating》提出了一种名为CUP（Comment UPdater）的自动化代码注释更新方法用来解决注释-代码不一致问题。CUP可以用来帮助开发人员执行JIT注释更新。当开发人员更改代码时，CUP可以自动为相关的注释提供更新建议。如果CUP生成的评论是正确的，开发人员可以通过一次点击快速执行评论更新。即使CUP的建议只是部分正确的，它们也可以减少开发者更新评论所需的编辑。因此，CUP可以帮助提高开发人员在JIT评论更新方面的生产力，并避免引入糟糕的评论。

相　关　方　法

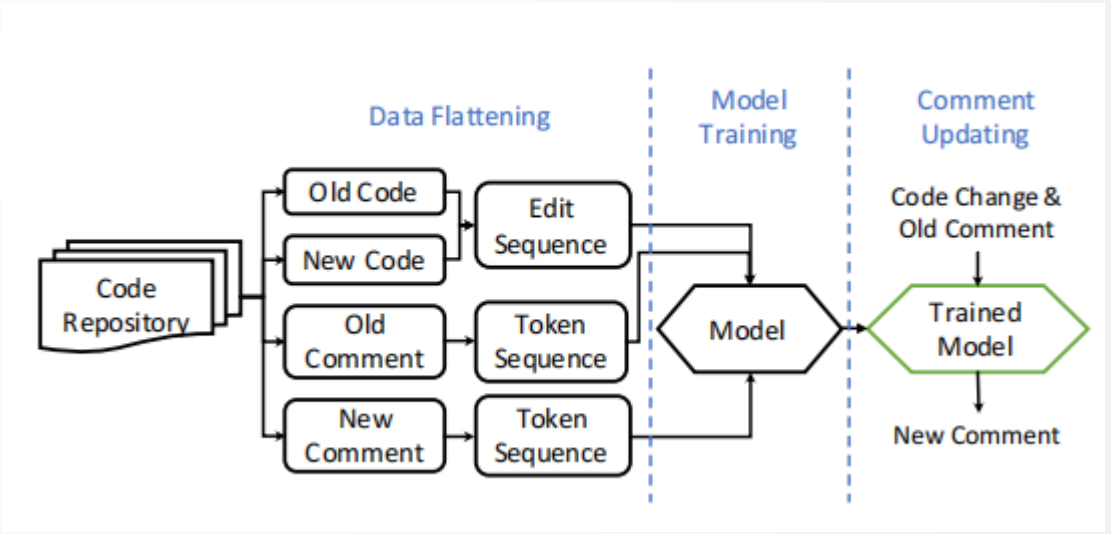
本文提出了一种名为CUP（Comment UPdater）的新方法，旨在自动化代码变更时的即时注释更新。该方法利用一种新颖的神经序列到序列（seq2seq）模型，从现有的代码-注释同时变更中学习注释更新的模式，并根据对应的旧注释和代码变更自动生成新的注释。CUP可用于辅助开发人员在代码变更过程中更新注释，从而减少和避免引入不良注释。

CUP总体框架如右图所示。它包括三个阶段，即数据扁平化、模型训练和注释更新。

具体来说，首先将从源代码存储库中提取的代码注释共更改样本扁平化为序列。

然后，利用扁平的数据训练一个神经seq2seq模型。

最后，给定一个代码更改及其关联的旧注释，经过训练的模型可以自动生成一个新注释来替换旧注释。



小　　结

作者构建了一个包含108K代码-注释共更改示例的数据集。用来评估CUP方法。实验结果表明，CUP在准确率和召回率等方面优于其他基线方法，并且能够在注释更新中显著减少开发人员的工作量。总的来说，CUP为自动化注释更新提供了一种有效的解决方案，有助于改善软件维护和开发过程中的注释质量。

数据集限制

作者的数据集仅来自Java项目，并且仅包含方法注释的更新。这可能限制了该方法在其他编程语言和注释类型上的适用性和泛化能力。

GumTree方法映射的局限性

作者使用GumTree进行方法匹配，但某些由GumTree识别的方法映射是次优的。尽管作者采取了一些措施来提高匹配的准确性，但仍可能存在方法映射不准确的情况。

Part three

启发式代码注释更新

Trust me

研　　究　　问　　题

这当代码发生变化时，开发人员经常忘记更新注释的情况。《Automated Comment Update: How Far are We?》介绍了一个名为CUP的方法，利用神经机器翻译的技术自动生成与代码变化相一致的最新注释。然而，尽管CUP表现出良好的性能，但其完成度尚未得到全面评估。因此，研究者对CUP的有效性进行了深入分析，在多个方面进行评估，并提出了一种基于启发式方法的注释更新方法（HEBCUP），以与CUP进行对比。他们的研究旨在揭示CUP的优点和局限性。

本文提出了一种基于启发式的注释更新方法HEBCUP，并将其与CUP方法进行对比。研究分析表明，CUP的整体有效性主要来自于通过单个标记更改成功更新注释的情况（96.6%）。然而，当CUP忽略了某些代码变化信息（10.4%）或被额外信息误导时（12.8%），会出现一些更新失败的情况。为了对比CUP的成果，研究团队实现了HEBCUP，这是一种基于启发式规则的简单方法用于代码注释更新。借鉴对CUP成功和失败案例的观察，研究团队设计了一些启发式规则，以便将注释更新聚焦于变化的代码并进行标记级别的注释更新。

本文提出了一种名为HEBCUP的方法，用作自动化注释更新的基准线。HEBCUP是基于启发式的注释更新方法，通过设计一系列基本的启发式规则来更新注释，以实现代码变更的注释自动更新。

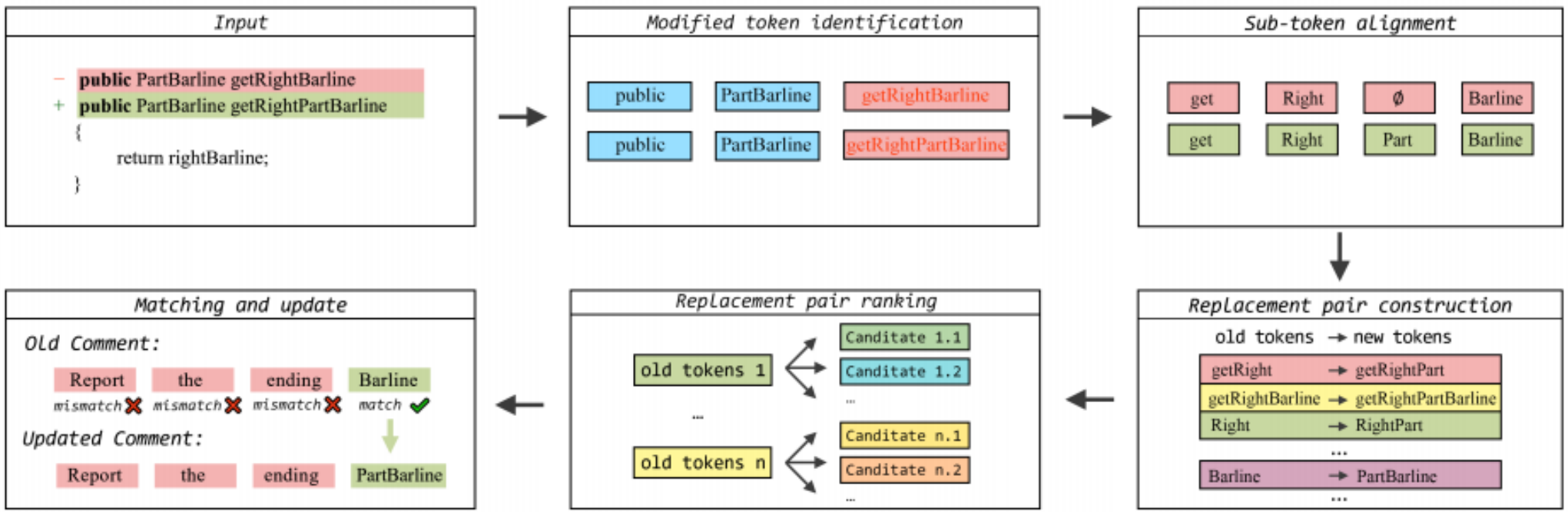
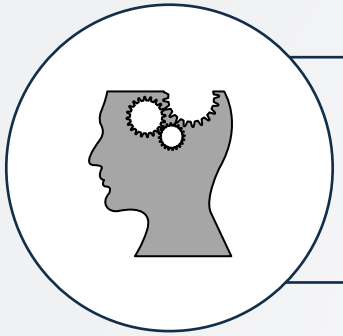


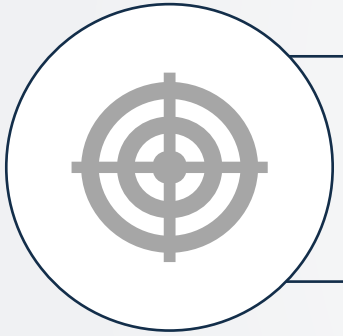
Fig. 9: Overview of the HEBCUP for comment update.

相　关　方　法



关注变化的代码

它通过分析代码变化的部分来确定是否需要更新注释。这样可以避免对没有发生变化的代码进行不必要的注释更新，提高了注释更新的效率。



令牌级别的注释更新

它通过对比代码变化前后的令牌，并根据变化的令牌来更新注释。这种精细的注释更新方式可以确保注释与代码的变化保持一致，提高了注释的准确性和可读性。

启发式规则

这些规则基于对CUP生成的注释进行实证分析得出，可以根据代码的特征和上下文信息来确定注释的更新策略。这些启发式规则简单而有效，使得HEBCUP的实现相对简单快速。

小　　结

作者对CUP进行了实证研究，并调查了它的工作原理和失败原因。现有方法在更新注释时主要通过对现有注释进行微小改动来生成新的注释，大部分情况下仅修改一个标记。当代码变化中的相关内容可以直接从代码中获取时，现有方法能够生成正确的注释，但它们经常忽略一些代码变化信息，同时也容易受到其他信息的误导，导致失败的情况。基于对现有方法成功和失败案例的观察，作者设计了一种基于启发式的注释更新方法（HEBCUP），该方法通过关注变化的代码和进行标记级别的注释更新，在准确性方面优于现有方法，并且速度快了三个数量级。

当然， HEBCUP仍旧存在一些问题，仅适用于代码指示性更新，HEBCUP方法只能处理代码中的指示性更新，对于那些更新内容未在代码更改中呈现的注释，无法提供有效的更新。这限制了该方法在处理更复杂或非指示性注释更新时的适用性。

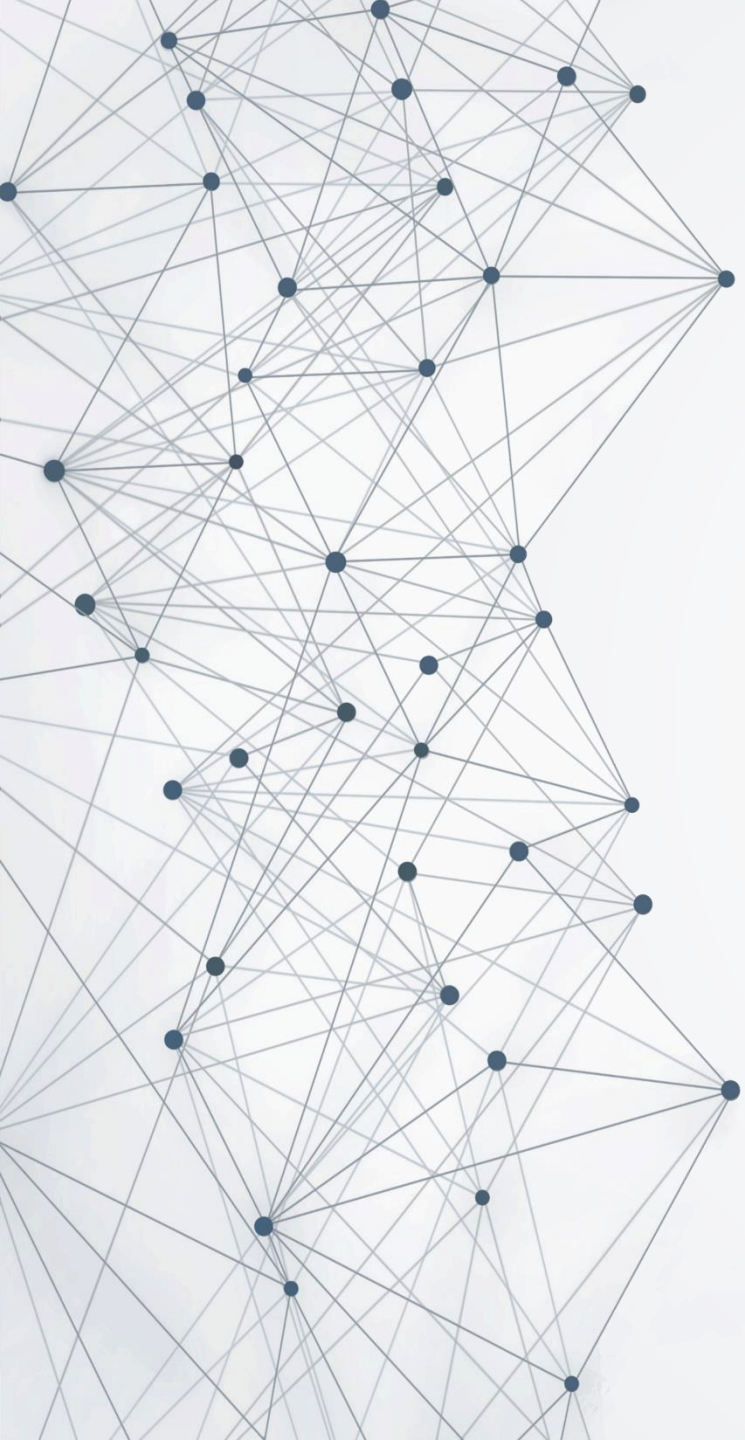
Part four

总结

Trust me

总　　结

这三篇关于代码注释更新的研究提供了有关自动化注释更新的深入洞察和方法。它们共同强调了代码注释在软件开发过程中的重要性以及代码演化过程中注释更新的挑战。这些研究表明，代码注释对于代码理解和维护至关重要，一个过时或者错误的注释往往会导致这份代码的可读性降低。然而，在实践中，开发人员常常忘记更新代码的同时更新注释，导致注释与代码不一致或过时。自动化注释更新的方法可以帮助解决这个问题，并减少开发人员手动更新注释的工作量。



感 谢 您 的 观 看

汇报人： 欧阳卓文

学号：1023040926