重新审视和改进检索增强深度断言生成

抽象单元测试验证被测单元的正确性，已成为软件开发过程中必不可少的活动。单元测试由一个测试前缀和一个测试预言(例如，断言)组成，前者驱动被测单元进入特定状态，后者指定在该状态下的行为。为了减少进行单元测试的人工工作量，Yu等人提出了一种集成方法(简称集成)，将信息检索(information retrieval, IR)与基于深度学习的方法相结合，为单元测试生成断言。尽管前景看好，但对于集成为什么或在哪里起作用或不起作用，仍然存在知识鸿沟。在本文中，我们描述了对整合有效性的深入分析。我们的分析表明:①集成的整体性能主要得益于其成功检索断言。②集成很难理解检索到的focal-test (focal-test包括一个测试前缀和一个被测试单元)和输入的focal-test之间的语义差异，导致许多token被错误修改;③集成仅限于特定类型的编辑操作(即替换)，无法处理token的添加或删除。为了提高断言生成的有效性，本文提出了一种新的检索和编辑方法EDITAS。具体来说，EDITAS首先从预定义的语料库中检索类似的焦点测试，并将其断言视为原型。然后，EDITAS重用原型中的信息，并自动编辑原型。相比集成，EDITAS的泛化性更强，因为它可以注释全面理解输入和类似焦点测试之间的语义差异;用更大的灵活性应用适当的断言编辑模式;生成更多样化的编辑操作，而不仅仅是替换操作。在两个大规模数据集上的实验结果表明，EDITAS的准确率和BLEU值平均提高了10.00% ~ 87.48%和3.30% ~ 42.65%。

*索引术语-单元*测试，断言生成，测试断言，深度学习

I. 介绍

单元测试是软件开发中至关重要的活动，它涉及对软件应用程序的单个单元进行测试，如方法、类或模块。当集成和系统测试评估系统的整体性能时，单元测试侧重于验证每个代码单元是否按照开发人员的预期和构想工作，从而在故障在整个系统中传播之前检测和诊断故障，并防止回归。有效的单元测试可以提高软件的质量，降低软件故障的发生率和成本[1]，[2]，以及增强整个软件开发过程。

单元测试由一个*测试前缀*(test prefix)和一个测试预言(*test oracle*)组成，前者是一系列操作被测单元以达到特定状态的语句，后者通常包括一个指定该状态下预期行为的断言(断言)[3]。在- - - - - -

```
1 @Test
2 public void test01() throws Throwable  {
3     CategoryAxis3D categoryAxis3D0 = new CategoryAxis3D();
4     categoryAxis3D0.setVisible(false);
5     CategoryAxis3D categoryAxis3D1 = new CategoryAxis3D();
6     boolean boolean0 = categoryAxis3D1.equals(categoryAxis3D0);
7     assertFalse(categoryAxis3D0.isVisible());
8     assertFalse(boolean0);
9 }
```

图1所示。单元测试用例的例子。

stance，在图1所示的单元测试中，第3-6行构成了测试前缀，它创建了两个CategoryAxis3D对象，将categoryAxis3D0设置为不可见;测试人员利用一个布尔变量来检查categoryAxis3D0和categoryAxis3D1是否相等。在第7-8行，断言指定了预期的结果，测试在执行测试前缀之后，categoryAxis3D0的可见性属性应该为False，并且这两个对象不应该相等。

尽管测试带来了显著的好处，但创建有效的单元测试是一项非琐碎且耗时的任务。之前的研究表明，开发人员可以花费超过15%的时间在测试生成[4]上。为了简化单元测试生成，各种自动化测试工具被提出，如Randoop[5]和EvoSuite[6]。然而，这些测试生成工具优先生成高覆盖率的测试，而不是有意义的断言，并且在理解预期的程序行为方面面临挑战。例如，EvoSuite生成的断言的工业评估[7]表明，“在手动编写的测试中，断言与生成的断言不同，是有意义和有用的。”因此，在进行单元测试时仍然需要大量的手工工作。

为了减少oracle生成的工作量，Watson等人引入了ATLAS[8]，这是一种基于深度学习(DL)的技术，在现有单元测试的广泛语料库上训练神经生成模型。ATLAS通过采取一对由测试前缀及其*焦点方法(即测试中的方法)*组成的操作，该方法既包含方法名称，也包含方法主体。为了与之前的研究[9]保持一致，我们将这样的一对称为*focal-test*。ATLAS然后从头开始为focal-test生成一个断言。与基于规范挖掘的方法不同，神经模型更加灵活，尤其是在文档不精确或不完整的情况下。因此，ATLAS为测试断言问题提供了一个更具适应性的解决方案。然而，ATLAS的有效性受到几个问题的限制:1)ATLAS一般更喜欢语料库中的高频词，可能在处理低频词时出现问题，例如

作为项目特定的标识符。2) ATLAS在生成长序列的token作为断言时性能较差。

最近，Yu等人[9]提出了一种基于信息检索(IR)的断言生成方法，包括基于IR的断言检索(IR_{ar})和检索-断言自适应(RAadapt)技术。IRar采用与ATLAS相同的输入，并基于Jaccard相似系数[10]检索与给定焦点测试最相似的断言。然后，RA_{adapt}根据上下文进一步调整检索出的断言中的token。此外，还提出了一种集成方法[9](缩写为*Integration)*，将基于ir的方法与基于dl的方法相结合，以提高断言生成能力。该集成方法验证了检索到的断言与当前focal-test之间的兼容性。如果兼容性超过阈值，则返回检索到的断言作为最终结果。否则，基于dl的方法生成断言。实验结果表明，与ATLAS相比，该集成方法取得了更高的准确率和BLEU分数。虽然性能在断言生成研究中是一个显著的成就，但仍然存在关于所提出的技术为什么或在哪里有效或无效的知识差距。

为了填补文献中的这一空白，我们首先对*集成*进行了全面的评估，以获得对其应用场景的更好理解。我们在Yu等人采用的两个公开数据集[9]($Data_{old}$和$Data_{new}$)上进行了实证评估。主要发现是:

- 集成主要依赖于基于ir的生成断言的方法。例如，*集成*利用基于ir的方法为Datanew生成80.06%的断言。此外，对于Dataold和Datanew，基于ir的方法生成的正确断言与检索出来的断言有83.38%和92.47%是相同的。
- 集成仅在其适应操作中替换标记，使其具有挑战性，以推广到复杂的断言生成场景，并限制其有效性。
- 集成经常错误地修改断言，即使检索到的断言与真实情况完全匹配。当一个token需要被修改以获得正确的结果时，*集成*的平均精度仅为20.14%，并且对于超过5个token下降到仅1.91%。
- 集成未能理解焦点测试之间的语义差异，导致在许多情况下检索到的断言需要修改，但它直接作为预期的断言返回。

总的来说，我们的实证研究重申了之前的发现，即基于焦点测试的相似性，我们总是可以找到“几乎正确”的断言，与正确的断言非常相似。然而，我们也确定了几个限制*集成的有效性和泛化性的限制*。一方面，单个token替换操作在复杂的断言编辑场景中挣扎，导致在修改五个以上token时*集成*的准确率低(1.9%)。另一方面，*集成*无法理解输入和类似焦点测试之间的语义差异，导致其做出不正确的编辑操作，或在必要时无法编辑检索到的断言。

为了缓解上述限制并获得更高的性能，提出了一种新的断言生成方法EDITAS。IR的有效性意味着类似焦点测试的断言可以被重用。换句话说，预期断言中的某些标记也极有可能出现在检索的断言中。基于规范挖掘和检索断言自适应方法的改进揭示了断言模式的重要性。受此启发，EDITAS将来自类似焦点测试的断言视为原型，并利用神经序列到序列模型来学习用于修改原型的断言编辑模式。我们的动机是，检索到的断言指导神经模型“如何断言”，断言编辑模式向神经模型突出“要断言什么”。

EDITAS由两个主要组件组成:检索组件和编辑组件。给定一个输入的焦点测试，检索组件从语料库中获得其相似的焦点测试，并利用检索到的焦点测试的相应断言作为原型。在编辑组件中，基于代表输入和相似焦点测试之间的语义差异的编辑序列，训练一个序列到序列的神经网络来编辑原型。一方面，EDITAS通过编辑断言而不是从头开始生成断言，有效缓解了冗长的断言生成问题，这是ATLAS的性能瓶颈。此外，EDITAS比*Integration*更具泛化性，因为它可以1)全面理解焦点测试之间的语义差异;2)应用适当的断言编辑模式，具有更大的灵活性;以及3)生成更多样化的编辑操作，而不仅仅是替换操作。

将基于ir的断言检索方法(IR_{ar})、检索到的断言自适应方法包括RAHadapt和RAadaptNN、*集成方法Integration*和ATLAS作为基线。在Dataold和Datanew数据集上对EDITAS和基线进行了准确性和BLEU方面的评估。评估结果表明，在断言生成方面，EDITAS在所有指标上都优于所有基线。具体来说，EDITAS达到了最高的准确率，在两个数据集上分别比基线提高了14.87%-70.15%和5.12%-104.80%。

综上所述，本文的贡献包括:

(1)对结合DL和IR技术的最先进的断言生成方法进行了深入分析。我们的分析结果为该方向的未来研究提供了有价值的见解。

(2)提出了一种新的检索和编辑方法，即EDITAS，用于断言生成。EDITAS利用来自类似焦点测试的断言作为原型，并使用神经序列到序列模型来学习断言编辑模式。

(3)进行了广泛的实验来评估我们的方法。实验结果表明，EDITAS显著优于所有基线。

(4)我们开源了我们的复制包[1]，包括数据集、EDITAS的源代码、训练好的模型和断言生成结果，以供后续研究。

II. 背景及相关工作

A. 基于dl的断言生成

随着深度学习(DL)的兴起，越来越多的软件工程任务可以利用先进的DL技术有效解决，如代码搜索[11]、[12]、程序自动修复[13]-[16]、故障诊断[17]、代码摘要[18]-[21]、代码克隆检测[22]-[26]等。Watson等人[8]最近提出了第一个基于dl的断言生成方法ATLAS。ATLAS利用神经机器翻译(NMT)为给定的focal-*test生成断言*，该断言由一个没有任何断言的测试方法(即测试前缀)和它的focal方法(即被测方法)组成，包括方法名称和方法主体。为了开发ATLAS, Watson等人首先从使用JUnit测试框架的GitHub项目中提取了测试-断言对(tap)。每一对都由一个焦点测试(focal-test)及其相应的断言组成。然后，初始的TAP set被预处理成两个数据集:1)原始源代码，其中TAP被简单标记化(见图2顶部)，2)抽象代码，其中不常见的token被进一步表示为原始源代码中各自的类型和顺序id(见图2底部)。最终，ATLAS生成一个有意义的断言来验证focal方法的正确性。

最近的研究[27]-[29]探索了预训练模型(如T5和BART)的潜力，通过预训练和微调来支持断言生成任务。具体来说，这些方法涉及使用大量源代码或英语语言语料库对转换模型进行预训练，然后对其进行微调以进行断言生成。Dinella[3]提出TOGA，通过在一组候选测试断言上使用排名架构来解决断言生成问题。TOGA利用语法和基于类型的约束来限制候选的生成空间，并使用基于transformer的神经方法来获得最有可能的断言。我们发现TOGA[30]，[31]的形式化实现需要一个种子/近似断言来确定用于构造和优化候选断言集的变量。此外，种子/近似断言需要手动创建或通过EvoSuite工具[6]创建。与TOGA不同，本文关注的是仅使用focal-test生成断言的问题。

B. 结合信息检索和深度学习进行断言生成

社区一直致力于通过利用信息检索(IR)技术来推动DL技术在软件工程任务中的应用，并取得了有希望的结果。借鉴

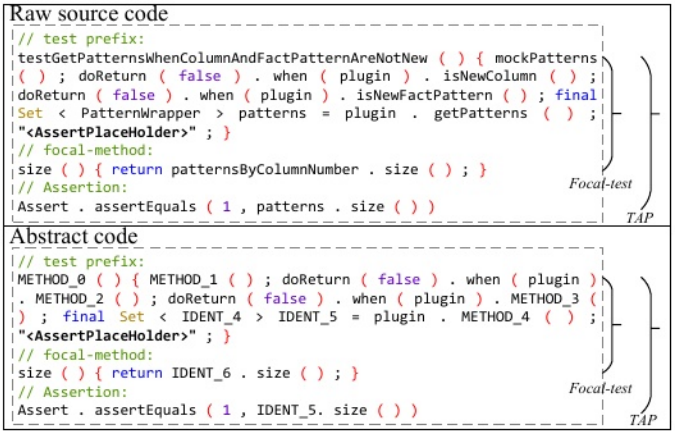

图2所示。抽象的过程。

“结合IR和DL”的想法，Yu等人[9]提出了一种新颖的集成方法来解决断言生成问题。他们的贡献包括两种基于ir的断言生成方法，即基于ir的断言检索(IRar)和检索的断言自适应(RAadapt)。

IR_{ar}的基本思想是检索其对应的focal-test与给定的focal-test具有最高相似度(例如，Jac-card[10]相似度)的断言，并将检索到的断言作为预期断言返回。由于IR_{ar}并不总是检索完全准确的断言，因此有人提出RA_{adapt}，利用上下文信息自动将检索到的断言适应正确的形式。对于检索到的断言，RA_{adapt}执行以下适应过程:1)决定是否应该修改断言;2)决定应该修改哪个令牌(即被调用的方法、变量或常量);3)决定候选令牌应该被替换为什么值。Yu等人提出了两种确定替换值的替换策略:一种基于启发式(RAHadapt)，另一种基于神经网络(RANNadapt)。RAadapt利用词法相似度进行代码替换。与RAHadapt相比，$RANN_{adapt}$通过使用神经网络架构合并语义信息，进一步增强了词汇相似性，并计算代码适配的替换值。最后，Yu et al.[9]结合了IR和DL技术，提出了一种集成的方法，本文称为集成。*Integration*首先使用Jaccard相似性检索断言，并在必要时调整检索到的断言。然后，*Integration*使用语义兼容性推理模型来计算调整后的断言和当前焦点测试的“兼容性”。如果兼容性低于指定阈值(记为t)，则*Integration*切换到ATLAS，从零开始生成一个断言。t的值是根据验证集确定的。鉴于RANNadapt优于其他基于ir的方法，包括IRar和RAadaptH，我们采用$RAadapt_{NNand}$ ATLAS的组合来探索集成的最优性能。

[1] https://anonymous.4open.science/r/EditAS-4CD6

III. 整合的实证探索

实证研究旨在调查集成[9]的应用场景，并深入了解其机制，即集成在哪里以及为什么有效和不有效。为此，设计了三个研究问题。

- RQ1:*数据集的特征是什么*?我们首先对数据集进行深入分析。特别是，我们分析了整个数据集上断言长度的分布。接下来，对于测试集中的每个测试-断言对TAP_i，我们从其对应的focus-test与TAP_i的focus-test相似度最高的训练集中检索断言，并计算TAP_i的断言与检索到的断言之间的编辑距离。这样的调查不仅验证了数据集的特征，而且还帮助我们理解tap和它们的相似实例之间的内在关联，这可以指导我们的方法设计(参见第四节)。
- RQ2:*集成在哪里以及为什么起作用*?我们进一步探讨了在断言生成方面集成的优势。我们首先根据使用的断言生成方法对集成生成的结果进行分类。接下来，我们分析正确生成的断言。
- RQ3:*集成在哪里以及为什么失败*?我们调查了*集成的弱点*，这是至关重要的，因为了解一种方法的应用场景可以更好地帮助我们在实践中应用[32]。

A. 数据集

我们使用Yu等人的两个公开可用的数据集[33]进行实验，分别是Dataold和Datanew。为了使我们的论文自成一体，我们在接下来的段落中简要描述了$Data_{old}$和$Data_{new}$。

1) Dataold: Dataold来源于ATLAS使用的原始数据集。最初，$Data_{old}$是从GitHub中250万个测试方法池中提取出来的，这些测试方法包括测试前缀和它们对应的断言语句。对于每个测试方法，$Data_{old}$都包含了focal方法，即测试中的生产代码。然后对$Data_{old}$进行预处理，以排除token长度超过1K的测试方法并进行过滤

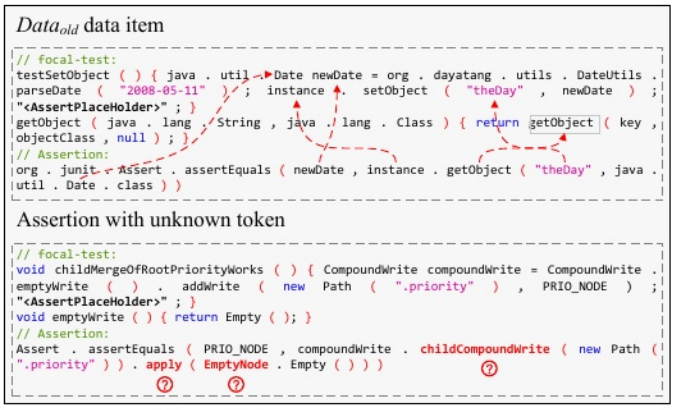

图3所示。使用已知标记*和*未知标记进行断言。

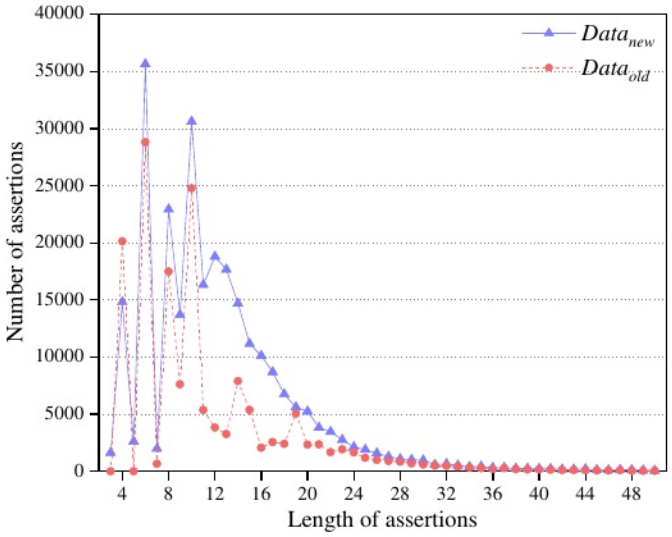

图4所示。每个数据集断言的长度分布。

包含焦点测试和词汇表中不存在的*未知*token的断言，遵循自然语言处理[34]，[35]的既定实践。例如，图3的底部突出显示了未知标记childCompoundWrite、apply 和EmptyNode。在去除重复项后，$Data_{old}$获得了156,760个数据项，这些数据项以8:1:1的比例被进一步划分为训练集、验证集和测试集。

2)$Data_{new}$:排除具有未知令牌的断言会过度简化断言生成问题，使$Data_{old}$不适合表示现实的数据分布。这反过来又对实验结论的有效性构成了重大威胁。因此，Yu等[9]通过在Dataold中添加那些被未知token排除的案例，构建了一个扩展数据集，记为$Data_{new}$。除了Dataold中已有的数据项之外，Datanew还额外包含了108,660个带有未知token的样本，总共形成265,420个数据项，这些数据项也以8:1:1的比例划分为训练集、验证集和测试集。

B. 研究结果

1) RQ1:数据集的特征是什么:图4显示了断言长度的分布，其中X轴代表各种断言长度，Y轴代表相应断言的数量。两个数据集都揭示了一个相似的分布趋势:断言通常小于30个token，并且随着断言长度的增加，断言的数量减少。在Datanew中断言的平均长度是13个token，而在Dataold中是12个token。这一发现表明，在实践中，测试人员可能更喜欢简短的断言，因为长断言可能会导致维护困难。

Datanew和Dataold在断言长度和编辑距离上都表现出长尾分布。大部分断言长度集中在少量令牌内。

表我
每个测试集检索到的断言和真实值之间的编辑距离(e)

Dataset	$E=0$	$E=1$	$E=2$	$E=3$	$3<E\leq 5$	$5<E\leq 10$	$10<E$	Total
$Data_{old}$	5,684	2,377	934	598	1,020	2,082	2,981	15,676
$Data_{new}$	10,059	3,059	1,581	1,031	1,563	3,050	6,199	26,542

表I进一步提供了测试样本的基本事实(即，预期的断言)与检索断言之间的编辑距离。有趣的是，我们可以观察到相当大比例的as-sertions与检索断言完全匹配，在Datanew和Dataold中分别占37.90% = 10,059 / 26,542和36.26% = 5,684 / 15,676。此外，对于Datanew和$Data_{old}$，65.15%和67.70%的测试样本只需要修改检索到的断言少于或等于5个token，就能产生正确的断言。这样的发现为信息检索(information retrieval, IR)方法的有效性提供了证据:通过从焦点测试中识别相似性，我们可以找到几乎或完全匹配的断言。

信息检索(IR)技术可以通过识别焦点测试的相似性成功地搜索接近或完全正确的断言。

*2) RQ2:集成在哪里和为什么工作:*为了深入了解*集成*在哪里和为什么工作，我们分析了其预测结果。考虑到*集成*是一种集成方法，集成了ATLAS和基于ir的断言生成方法(本文中是RANNadapt)，我们根据使用的断言生成方法对*集成*生成的结果进行分类。我们的分析揭示了RA^{NN}

适应

为$Data_{new}$生成21,250(80.06% = 21,250 / 26,542)断言，为$Data_{old}$生成10,215(65.16% = 10,215 / 15,676)断言。结果表明，*集成*偏好基于ir方法生成的断言。

*集成*的有效性主要依赖于基于ir的断言生成方法。

由于基于ir的方法对*集成贡献最大*，我们对RANNadapt生成的正确断言进行了进一步的分析。图II显示了检索到的断言与RANNadapt为Datanew和$Data_{old}$生成的正确断言之间的编辑距离。从图中可以看出，RANNadapt的成功主要是基于检索与ground truth相同的断言，如在$Data_{new}$中E = 0的样本占92.47% = 9,839 / 10,640。

表二世
检索到的断言与RANNadapt生成的断言之间的编辑距离(E)

Dataset	Prediction	$E=0$	$E=1$	$E=2$	$E=3$	$E=4$	$E=5$	$E>5$	Total
$Data_{old}$	*Correct*	5,435	486	341	108	73	35	40	6,518
	Incorrect	52	1,453	342	255	224	226	1,145	3,697
$Data_{new}$	*Correct*	9,839	437	235	67	32	8	22	10,640
	Incorrect	130	2,436	1,162	798	709	543	4,832	10,610

RANNadapt通过适配操作增强了*集成*的断言生成，但对于单个或两个token修改比其他修改情况更有效。例如，使用Datanew数据集，RANNadapt在仅修改一个token时达到了15.21% = 437/(437 + 2,436)的准确率，在修改三个token时达到了7.75% = 67/(67+798)的准确率。

RANNadapt产生的大多数成功断言与检索到的断言完全相同。在适应操作方面，RANNadapt在单个token修改上的表现优于其他修改情况。

3) RQ3:集成失败的地方和原因:之前的工作[9]已经证明，ATLAS性能的瓶颈之一来自于用长序列生成断言。因此，在RQ3中，我们专注于探索RANNadapt在哪里以及为什么失败。我们首先收集了由RANNadapt生成的不正确的断言，并计算检索到的断言与它们的真实值之间的编辑距离。如表II所示，对于$Data_{new}$，有130个测试样本的断言与ground truth匹配，但仍然被RANNadapt修改，在Dataold中也发生了类似的现象。RANNadapt的适应策略的表现是有限的。例如，当E = 1时，RANNadapt错误地为$Data_{new}$生成84.79% = 2,436 /(437+ 2,436)的断言，为Dataold生成74.94% = 1,453 /(1,453 +486)的断言。当E >为5时，RANNadapt的性能更差，平均准确率仅为1.91%。

*集成*经常错误地修改断言

即使检索到的断言与地面匹配
真理究竟。当修改一个token来获取
正确的断言，平均准确率仅为20.14%。

表3
RANNadapt针对不正确的断言所做的编辑数(N)

Dataset	$N=0$	$N=1$	$N=2$	$N=3$	$N=4$	$N=5$	$N>5$	Total
$Data_{old}$	3,071	244	173	90	51	26	42	3,697
$Data_{new}$	9,436	497	328	167	73	35	74	10,610

我们进一步计算了RANNadapt为那些不正确的断言所做的编辑数。如表III所述，我们可以观察到，有很大比例的样本RANNadapt没有对其检索断言进行任何更改，例如在$Data_{new}$中，这种类型的样本量为9436，占错误断言总数的88.93%。我们的分析认为，这可能是由于RANNadapt在理解焦点测试之间的语义差异方面的困难。Yu et al.[9]认为，如果一个断言包含至少一个从输入的焦点测试中缺失的token，那么它需要被编辑。然而，这种设置忽略了许多需要进行必要更改以适应输入焦点测试上下文的实例，即使检索到的断言的所有标记都发生在输入焦点测试中。

RQ3的查找-2

*集成*未能理解焦点测试之间的语义差异，导致在许多情况下，检索到的断言需要修改，但没有适当编辑。

IV. 检索和编辑断言生成

根据从实证研究中获得的见解，本文提出EDITAS，一种用于生成断言的检索和编辑方法。与仅考虑替换操作以调整检索断言的*集成*不同，EDITAS可以从类似的点击中学习不同的断言编辑模式。给定一个focal-test作为输入，EDITAS根据focal-test的相似性检索语料库以获得相似的TAP，并通过检索到的TAP的断言和兼容的编辑模式生成一个新的断言。EDITAS的整体框架如图5所示。EDITAS由一个检索组件和一个编辑组件组成。

A. 检索组件

在我们的方法中，检索组件从给定输入焦点测试的语料库中检索类似的测试-断言对(TAP)。具体来说，为了促进高效检索，EDITAS首先使用javalang[36]对训练和测试集中的每个焦点测试进行分词，并删除重复的token。然后，检索组件检索其焦点测试与输入焦点测试具有最高杰卡德相似系数的TAP。Jaccard相似度是一种考虑两个文本之间单词重叠的文本相似度度量，使用以下公式计算:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

其中A和B是两个词袋，|·|表示集合中元素的数量。

B. 编辑组件

采用神经编辑模型从相似的tap中学习不同的断言编辑模式。编辑模型旨在理解如何根据焦点测试之间的语义差异将一个断言修改为另一个断言。具体来说，对于给定的焦点测试ft及其类似的焦点测试实例ft′，以及它们相应的断言x和y，神经编辑模型旨在找到一个函数f，使f (ft, ft′, x) = y。在本节中，我们详细阐述焦点测试语义差异表示和神经编辑模型训练。

*1)聚焦测试语义差异表示:*我们使用编辑序列提取并比较聚焦测试之间的语义信息和修改细节，根据之前工作的finding[37]:两种方法之间的不同单词可以反映它们的语义差异。我们遵循与[38]、[39]类似的方法，使用diff工具对齐两个标记化的焦点测试序列，并基于所得到的对齐创建一个编辑序列。如图6所示，编辑序列中的每个元素(命名为*编辑*)被表示为三⊠ft_i，ft_i'，a_i⊠，其中ft_i是一个focal-test中的token，ft_i'is是类似的focal-test中的token，a_i是将ft_i转换为ft_i'的编辑动作。有四种类型的编辑动作:*插入*、*删除*、相等或*替换*。当a_i是*插入*(删除)操作时，意味着ft_i(ft_i')会是一个空token∅。构建这样的编辑序列，不仅可以保留focal-test(即ft_i和ft_i'))的信息，还可以通过ai突出它们的细粒度差异。

*2)神经编辑模型训练:*我们的神经编辑模型从根本上来说是一个序列到序列的神经架构。它接受一个检索断言x = ⊠xhD· E 1D 2, x，⋯，x| x| Ei和一个编辑sequence E =ft1, ft1, a1，⋯Ftn, ft 'n，一个as输入，被设计用来生成一个新的断言y = ⊠y_1，y_2，⋯，$y_{|y|}$。具体来说，EDITAS包含两个编码器:编辑序列编码器和断言编码器，分别对编辑序列和检索到的断言进行编码。然后在编码器端应用注意力机制来学习编辑序列和检索断言之间的关系。解码器由两个指针生成器组成，使模型能够在生成过程中同时从输入的焦点测试和检索到的断言中复制标记。这种方法可以有效地保留检索到的断言的原始含义，同时集成在编辑序列中反映的断言编辑模式。

❶编码器。两个编码器的结构，即编辑序列编码器和断言编码器，几乎完全相同。它们由上下文嵌入层、注意力层和建模层组成。

*上下文嵌入层。*我们首先将焦点测试标记、断言标记和编辑动作标记映射到嵌入。考虑到只有四个编辑动作，我们随机初始化一个嵌入矩阵，并在训练过程中更新它。为了同时捕获语法和语义信息，我们采用了一个预训练模型，如fastText[40]，来为每个焦点测试和断言token获取词嵌入。然后，我们使用双向长短期记忆(Bi-LSTM)[41]来处理词嵌入序列，以访问上下文信息。对于编辑序列编码器和每个编辑E_i，三个嵌入，即efti, efti '， eai首先被水平连接，然后输入到Bi-LSTM中，如下所示:

$$e_i' = [e_{ft_i} \oplus e_{ft_i'} \oplus e_{a_i}]$$
$$\overrightarrow{h}_i' = \mathrm{LSTM}(\overrightarrow{h}_{i-1}', e_i'); \overleftarrow{h}_i' = \mathrm{LSTM}(\overleftarrow{h}_{i+1}', e_i')$$
$$h_i' = [\overrightarrow{h}_i' \oplus \overleftarrow{h}_i']$$

其中h’i是这个编辑的上下文向量，⊕是一个拼接操作。类似地，断言编码器以x_i’s嵌入e_{x_i}作为输入获得每个断言令牌x_i的上下文向量h_i。

*注意力层。*这个层负责连接和融合焦点测试差异的信息和检索到的断言的信息，捕捉它们的关系。我们把它放在两个上下文嵌入层的顶部。attention层将上下文向量，即H和H '作为输入，

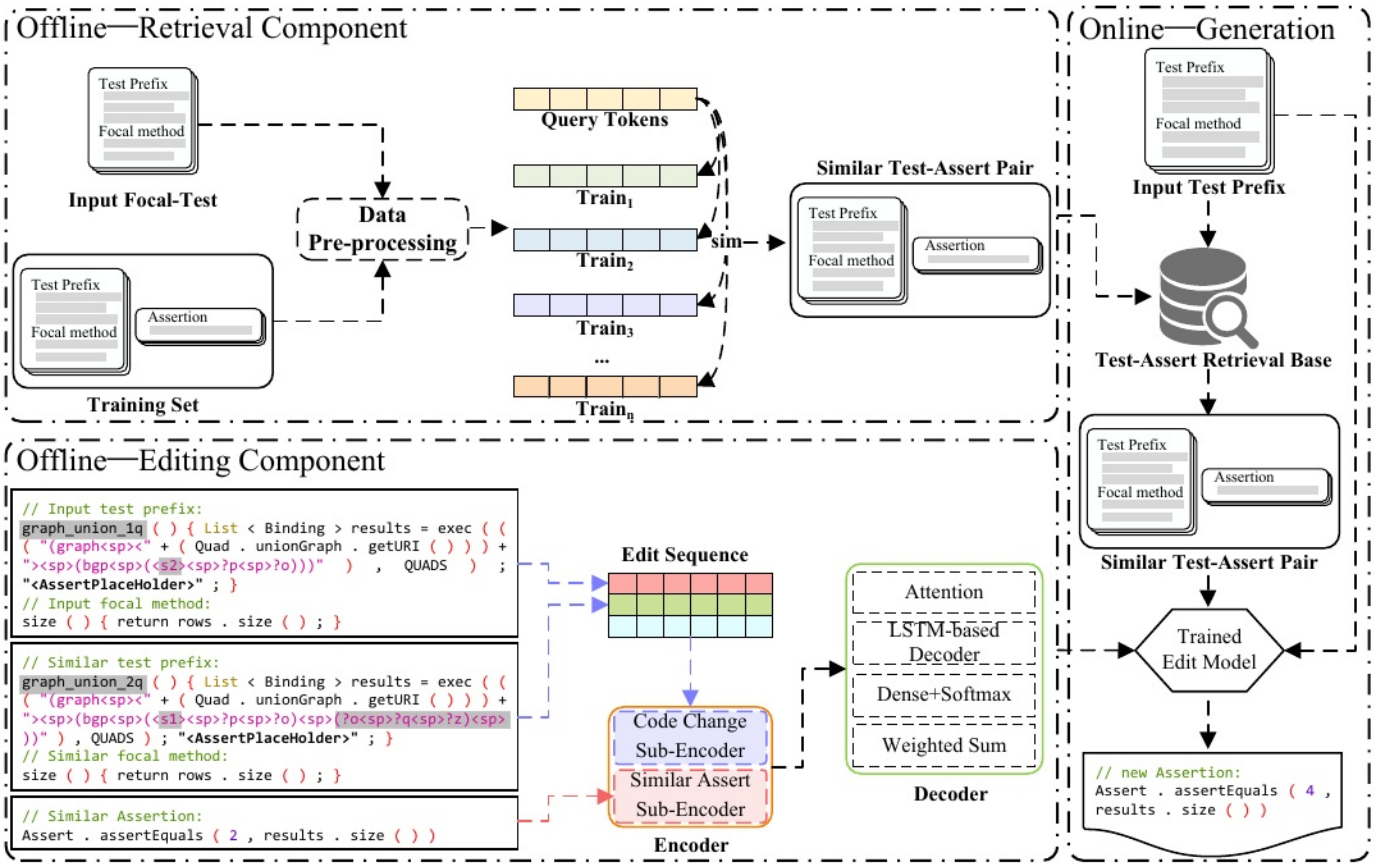

图5所示。我们方法的整体框架。

并输出每个编辑的断言感知(编辑感知)特征向量(断言令牌)，以及此编辑的原始上下文向量(断言令牌)。形式上，使用点生成注意力机制[42]计算断言令牌x_i的编辑感知特征向量g_i如下所示:

$$g_i = H^{'}\alpha_i; \quad \alpha_i = \text{softmax}(H^{'\top} W_\alpha h_i) \tag{2}$$

衡量每个编辑重要性的注意力权重α_i是关于toxi的。编辑Ei的断言感知特征向量g_i的计算几乎相同，可以表示为:

$$g_i^{'} = H\alpha_i^{'}; \quad \alpha_i^{'} = \text{softmax}(H^{\top} W_\alpha^{\top} h_i^{'})$$

建模层。该层使用两个不同的Bi-LSTM，分别基于每个编辑(断言令牌)的上下文向量和断言感知(编辑感知)特征向量生成最终的特征表示。为

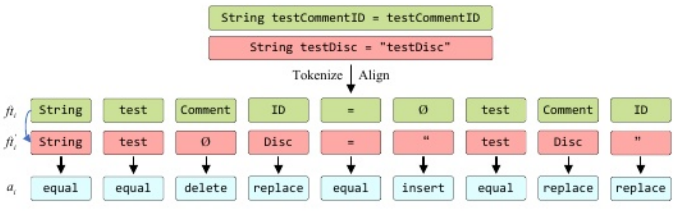

图6所示。将焦点测试之间的差异转换为编辑序列。

例如，给定一个断言令牌x_i，其最终表示z_i的计算如下:

$$f_i = [g_i \oplus h_i]$$
$$\overrightarrow{z}_i = \text{LSTM}(\overrightarrow{z}_{i-1}, f_i); \overleftarrow{z}_i = \text{LSTM}(\overleftarrow{z}_{i+1}, f_i)$$
$$z_i = [\overrightarrow{z}_i \oplus \overleftarrow{z}_i]$$

❷解码器。EDITAS使用一个基于lstm的解码器，从两个编码器Z的输出中生成一个新的断言

Z。两个建模层的最终隐藏状态被串联起来，并用作解码器的LSTM的初始状态。在解码第j步时，解码器根据ground truth断言$e_{y\boxtimes i}$的第j个词嵌入计算出隐藏状态s_j，前一个隐藏状态sj−1，以及前一个输出向量o_{j-1}，如下所示:

$$s_j = \text{LSTM}(s_{j-1}, [e_{\hat{y}_j} \oplus o_{j-1}]) \tag{3}$$

我们在每个时间步计算一个上下文向量，作为点积注意力机制对编码器输入的表示，如下式2。给定有两个编码器，解码器获得two上下文向量，即从检索的断言中获得cj，从焦点测试差异中获得c_j。使用cj、c’j和sj计算输出向量oj，使用softmax层获得相应的词汇分布P_j^{vocba}。

$$o_j = tanh(V_c[c_j \oplus c_j^{'} \oplus s_j]; \quad P_j^{vocba} = \text{softmax}(V_c^{'} o_j)$$

表4
每种类型在Datanew和Dataold的详细统计

AssertType	Total	Equals	True	That	NotNull	False	Null	ArrayEquals	Same	Other
$Data_{old}$	15,676	7,866 (50%)	2,783 (18%)	1,441 (9%)	1,162 (7%)	1,006 (6%)	798 (5%)	307 (2%)	311 (2%)	2 (0%)
$Data_{new}$	26,542	12,557 (47%)	3,652 (14%)	3,532 (13%)	1,284 (5%)	1,071 (4%)	735 (3%)	362 (1%)	319 (1%)	3,030 (11%)

其中V_c和$V_c^{'}$are可训练参数。P_j^{vocba}记录了每个生成token的概率，其中概率最高的元素将是解码步骤j下的输出。由于焦点测试的相似性，可以合理地假设新断言中的某些token也应该出现在检索的断言中，而在检索的断言中不存在的其他token应该包括在输入的焦点测试中。因此，我们采用指针生成器从检索到的断言和输入的focal-test中复制token:

$$P_j^{ass}(y_j) = \sum_{k:x_k=y_j} \beta_{jk}; \quad P_j^{ft}(y_j) = \sum_{k:t_k^{'}=y_j} \beta_{jk}^{'}$$

Pjass(yj)和Pjft(yj)分别是从检索到的断言和输入的focal-test中复制yj的概率。

step βjk和j，计算出的βjk是基于对xk和cj和Ek在c_{j-1}的权重上下文向量的注意力

y_j在时间步j处的条件概率则是Pjvocba、Pjass和Pjft的组合，即

$$p(y_j|y_{<j}, x, E) = \gamma_j P_j^{vocba}(y_j) + (1-\gamma_j)(\theta_j P_j^{ass}(y_j) + (1-\theta_j) P_j^{ft}(y_j))$$

γ_j和θ_j分别表示通过从词汇表中选择和从检索到的断言中复制生成y_j的概率。这两种概率都由单层前馈神经网络建模，该网络与解码器联合训练。

C. 代

给定一个输入焦点测试功能测试，EDITAS通过三个步骤生成一个断言:

*第一步:选择一个相似的点击。*EDITAS利用大规模训练数据集作为检索语料库。然后，检索组件从语料库中检索焦点测试与ft密切匹配的TAP。关于检索过程的进一步信息概述在第IV-A节。

*第二步:捕获焦点测试之间的细粒度语义差异。*通过步骤1,EDITAS获得了与ft相似的焦点测试及其相应的断言语句。EDITAS计算编辑序列以捕获ft和检索到的focal-test之间的语义差异。这部分的细节在第四- b节中描述。

*步骤3:将检索到的断言与焦点测试之间的语义差异*相结合。最后，训练好的神经编辑模型根据编辑序列调整检索到的断言，从而生成一个与ft相对应的断言。关于该模型的进一步细节可以在第IV-B节中找到。

V. 实验装置

在本节中，我们将描述实验中使用的数据集和评估指标。我们通过实验来回答以下研究问题:

- RQ4:与最先进的断言生成基线相比，所提出的EDITAS的表现如何?
- RQ5:不同的相似系数是否会影响EDITAS的性能?

A. 基线

为了回答上述研究问题，我们将EDITAS与五条基线进行比较。我们首先选择了ATLAS，这是第一个也是最经典的基于神经网络的断言生成方法。ATLAS利用序列到序列模型从头开始生成断言。鉴于EDITAS旨在重新审视和改进检索增强的深度断言生成方法，本文进一步采用了三种最先进的(SOTA)基于检索的方法，包括IRar、RAHadapt和RANNadapt。最后，我们提供了EDITAS和一种SOTA检索增强的深度断言生成方法*integration*之间的性能比较。有关更多细节，请参阅第二节。

B. 数据集

我们利用Dataold和Datanew来评估EDITAS和基线方法的有效性，遵循Yu等人的[9]。与Dataold相比，Datanew将被排除的带有未知token的案例重新添加到$Data_{old}$。详情请参考III-A节。表IV提供了两个数据集的测试集的详细统计信息，包括它们在不同断言类型中的分布情况。

C. 指标

与之前的工作[8]，[9]一致，我们在实验中使用了以下指标。

*1)准确性:*我们使用准确性来评估断言生成技术的有效性。具体来说，当且仅当生成的断言与真实情况完全匹配时，才被认为是准确的。准确性决定了生成的输出与预期输出在语法上匹配的样本的百分比。

*2) BLEU:*与之前的研究[8]、[9]一致，我们使用multi - BLEU来衡量生成的断言和真实值之间的相似性。BLEU计算候选序列(即生成的断言)对参考序列(即地面真值)的修正n-gram精度，其中n范围为1到4。然后对修改后的n-gram精度值进行平均，并对过短的句子施加惩罚。

表五我们的方法与每个基线的比较

Approach	$Data_{old}$		$Data_{new}$	
	Accuracy	BLEU	Accuracy	BLEU
ATLAS	31.42 (↑ 70.15%)	68.51 (↑ 17.90%)	21.66 (↑ 104.80%)	37.91 (↑ 67.40%)
IR_{ar}	36.26 (↑ 47.43%)	71.48 (↑ 13.00%)	37.90 (↑ 17.04%)	57.98 (↑ 9.45%)
RA^{H}_{adapt}	40.97 (↑ 30.49%)	73.28 (↑ 10.22%)	39.65 (↑ 11.88%)	59.81 (↑ 6.10%)
RA^{NN}_{adapt}	43.63 (↑ 22.53%)	73.95 (↑ 9.22%)	40.53 (↑ 9.45%)	59.81 (↑ 6.10%)
Integration	46.54 (↑ 14.87%)	78.86 (↑ 2.42%)	42.20 (↑ 5.12%)	60.92 (↑ 4.17%)
EDITAS	**53.46**	**80.77**	**44.36**	**63.46**

↑表示相对于最先进基线的性能提升

表六世

我们的方法的详细统计数据和每种断言类型的每个基线

Dataset	Approach	Total	AssertType								
			Equals	True	That	NotNull	False	Null	ArrayEquals	Same	Other
$Data_{old}$	ATLAS	4,925 (31%)	2,501 (32%)	966 (35%)	248 (17%)	598 (51%)	229 (23%)	236 (30%)	100 (33%)	47 (15%)	0 (0%)
	IR_{ar}	5,684 (36%)	2,957 (38%)	1,039 (37%)	449 (31%)	439 (38%)	314 (31%)	285 (36%)	111 (36%)	89 (29%)	**1 (50%)**
	RA^{H}_{adapt}	6,423 (41%)	3,300 (42%)	1,151 (41%)	536 (37%)	553 (48%)	335 (33%)	316 (40%)	120 (39%)	111 (36%)	**1 (50%)**
	RA^{NN}_{adapt}	6,839 (44%)	3,509 (45%)	1,225 (44%)	**551 (38%)**	610 (52%)	342 (34%)	341 (43%)	134 (44%)	126 (41%)	**1 (50%)**
	Integration	7,295 (47%)	3,714 (47%)	1,333 (48%)	546 (38%)	724 (62%)	348 (35%)	352 (44%)	148 (48%)	**129 (41%)**	**1 (50%)**
	EDITAS	**8,380 (53%)**	**4,131 (53%)**	**1,581 (57%)**	526 (36%)	**807 (69%)**	**577 (57%)**	**469 (59%)**	**167 (54%)**	122 (39%)	0 (0%)
$Data_{new}$	ATLAS	5,749 (22%)	2,900 (23%)	619 (17%)	537 (15%)	388 (30%)	126 (12%)	85 (12%)	47 (13%)	37 (12%)	1,010 (33%)
	IR_{ar}	10,059 (38%)	4,664 (37%)	1,436 (39%)	1,070 (30%)	600 (47%)	394 (37%)	286 (39%)	147 (41%)	113 (35%)	1,349 (45%)
	RA^{H}_{adapt}	10,525 (40%)	4,882 (39%)	1,487 (41%)	1,142 (32%)	651 (51%)	403 (38%)	297 (40%)	154 (43%)	121 (38%)	1,388 (46%)
	RA^{NN}_{adapt}	10,758 (41%)	4,988 (40%)	1,526 (42%)	1,161 (33%)	691 (54%)	401 (37%)	308 (42%)	162 (45%)	126 (39%)	1,395 (46%)
	Integration	11,201 (42%)	5,248 (42%)	1,566 (43%)	1,196 (34%)	711 (55%)	401 (37%)	313 (43%)	162 (45%)	128 (40%)	**1,476 (49%)**
	EDITAS	**11,773 (44%)**	**5,339 (42%)**	**1,702 (47%)**	**1,304 (37%)**	**800 (62%)**	**523 (49%)**	**376 (51%)**	**172 (47%)**	**139 (44%)**	1,418 (47%)

VI. 实验结果

A. RQ4: EDITAS *vs. Baselines*

EDITAS的*整体有效性*。我们计算了不同方法生成的断言和人工编写的断言之间的准确性和BLEU分数。实验结果如表5所示。我们注意到，ATLAS在所有方法中表现最差。这主要归因于两个原因:1)ATLAS作为典型的序列到序列深度学习模型，存在曝光偏差和梯度消失问题，导致生成长token序列作为断言的有效性较差。2) ATLAS生成包含未知token的语句的能力较弱，这显著降低了其整体性能。IR_{ar}从语料库中检索断言并将其作为输出结果，取得了比ATLAS更好的性能。这表明类似的focal-test的断言包含了一些有价值和可重用的信息，这也证明了我们使用类似的focal-test的断言作为原型是合理的。RAadapt Hand RAadapt nn进一步调整检索到的断言，以增强基于ir的方法生成断言的能力。然而，如表V所示，两种RAadapt Hand RAadapt NNis的自适应操作性能有限，特别是对于复杂的数据集。例如，与IR_{ar}相比，RAadapt nnn可以提高20.33%的准确率，而对于$Data_{new}$，提高仅为6.94%。类似的观察结果也适用于raadapt。*Integration*结合了IR和DL技术，取得了比ATLAS和基于IR的断言生成方法更好的准确性和BLEU分数。

从表V中可以看出，EDITAS相对于ATLAS实现了显著的性能提升，在两个数据集上的平均准确率提升了87.48%，BLEU分数提升了42.65%。这归功于EDITAS使用了检索到的断言中丰富的语义信息，而不是从头开始生成断言。我们的方法EDITAS优于基于ir的基线*方法和所有*评估指标的集成。具体而言，与IRar、RAHadapt、RANNadapt和*Integration相比*，EDITAS分别实现了32.24%、21.19%、15.99%和10.00%的平均准确率提升，这证明了所提出编辑模块的有效性。与基于ir的基线相比，EDITAS采用检索到的断言作为原型，并通过考虑输入和相似焦点测试之间的语义差异进行修改。通过结合神经网络和基于ir的方法的优势，EDITAS取得了最好的性能。

对不同断言类型的有效性。我们进一步比较了EDITAS和baseline方法对不同断言类型的有效性。表VI中的每一列代表一种断言类型，每个单元格在括号中显示正确生成断言的数量及其对应的比率。结果表明，对于几乎所有断言类型，特别是对于标准的JUnit断言类型，EDITAS的表现比基线方法要好。对于另一种断言类型，EDITAS的表现略逊于*Integration*。这可能是因为训练集有大量JUnit测试框架的样本，导致EDITAS学习更多JUnit的语法特征。总的来说，实验结果可以表明EDITAS在生成不同类型断言方面的通用性。

为了更好地理解为什么EDITAS优于RAHadapt和RANNadapt，我们手动检查了断言生成

结果。分析表明，与其他方法相比，EDITAS具有以下优势:1)EDITAS能够学习和应用不同的断言编辑模式，而RAHadapt和RAadapt nn无法处理token的添加或删除操作。2) RAHadapt和RAadapt nn仅当检索到的断言包含至少一个在输入焦点测试中不存在的token时才会修改它。然而，即使检索到的断言中的所有标记都出现在输入焦点测试中，由于焦点测试之间的语义差异，它仍然可能需要修改。相比之下，EDITAS利用概率模型从现有的焦点测试的语义差异中学习断言编辑的常见模式。总的来说，EDITAS学习的编辑模式更加多样化，可以覆盖更广泛的样本范围。

EDITAS在准确率和BLEU上明显优于所有基线方法，在两个数据集上的平均性能提升分别为10.00%-87.48%和3.30%-42.65%。

B. RQ5:不同相似系数下的有效性

EDITAS使用Jaccard作为默认的相似系数。在这个RQ中，我们的目的是检查相似性系数是否影响EDITAS的有效性。具体来说，我们开发了两个附加版本的EDITAS，分别利用两个常用的相似系数Dice[43]和Overlap[44]。给定两个集合X和Y, Dice和Overlap计算相似度如下。

$$DSC(X,Y) = |X \cap Y|/(|X| + |Y|)$$
$$Overlap(X,Y) = |X \cap Y|/min(|X|, |Y|)$$

表VII展示了使用不同相似度系数的EDITAS在两个数据集上的准确性。我们的结果表明，相似系数确实对EDITAS的有效性有影响。具体来说，Jaccard和DICE相似系数对EDITAS的有效性影响不大。值得注意的是，我们观察到，与Jaccard和DICE相比，重叠产生了最糟糕的准确性和BLEU分数。我们将其归因于重叠只考虑了两个焦点测试之间的重叠程度，而忽略了它们的差异。

表vii不同相似系数的表现

Dataset	Metrics	Jaccard	Overlap	DICE
$Data_{old}$	Accuracy	53.46	46.13	53.46
	BLEU	80.77	75.42	80.77
$Data_{new}$	Accuracy	44.36	38.22	44.24
	BLEU	63.46	56.37	63.59

RQ5总结

相似系数确实对EDITAS的有效性有影响。提高检索组件的能力可以导致EDITAS更好的断言生成性能。

VII. 有效性威胁

我们的方法的有效性面临三个主要威胁。首先，继之前的工作[8]、[9]之后，我们只在两个Java数据集上进行了实验。虽然Java可能不能代表所有编程语言，但我们实验中使用的数据集足够大，并提供足够的安全性来证明EDITAS的有效性。此外，EDITAS独家采用了与语言无关的特性，即编辑序列，并可应用于其他编程语言。其次，检索组件根据词汇相似性检索相似的焦点测试。这可能导致检索的焦点测试和输入的焦点测试仅在词汇级别上相似，同时表现出不同的断言。为了应对这一威胁，我们使用了一个大规模的Java数据集(247M)来增加检索语料库的规模和多样性。我们还提出了一个编辑组件，通过考虑输入的focal-test和检索到的focal-test之间的语义差异来修改原型，以缓解这种威胁。最后的主要威胁来自于EDITAS在生成可编译断言方面缺乏与其他现有断言生成方法的比较。虽然可编译性可以作为断言质量的另一项指标，但自动化构建一直是一项具有挑战性的任务，高度依赖于外部/内部设置/资源[45]，[46]。因此，自动编译大规模断言仍然是一个障碍。为了确保可扩展性，我们不使用可编译性和bug检测作为指标。然而，我们确实会检查由EDITAS生成的断言的语法，并验证它们是否都满足语法要求。这些结果是意料之中的，因为我们的检索组件可以保证生成的断言的语法正确性。

VIII. 结论

在本文中，我们重申了之前的发现，即信息检索(IR)总是可以找到一个与预期非常相似的“几乎正确”断言，并强调了之前方法在修改检索断言方面的缺点。为了缓解这些问题，本文提出了一种名为EDITAS的用于断言生成的新型检索和编辑方法。EDITAS包含两个组件。一个检索组件，用于检索类似的focal-test，该组件由一个没有任何断言的测试方法及其焦点方法(即，被测试的方法)组成。编辑组件将相似焦点测试的断言视为一个原型，并结合输入焦点测试和相似焦点测试之间的语义差异所反映的原型和断言编辑模式来生成目标断言。我们在两个大规模的Java数据集上进行了广泛的实验。实验结果表明，EDITAS的性能大大优于最先进的基线。我们的工作表明，对于断言生成任务，检索相似的断言并通过应用一组编辑操作来学习修改检索的断言可以取得令人满意的性能。我们的源代码和实验数据可以在https://anonymous.4open.science/r/EditAS-4CD6上找到。

参考文献

[1] A. Hartman, "Is ISSTA research relevant to industry?" in *Proceedings of the International Symposium on Software Testing and Analysis, ISSTA 2002, Roma, Italy, July 22-24, 2002*. ACM, 2002, pp. 205–206. [Online]. Available: https://doi.org/10.1145/566172.566207

[2] S. Planning, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology*, vol. 1, 2002.

[3] E. Dinella, G. Ryan, T. Mytkowicz, and S. K. Lahiri, "TOGA: A neural method for test oracle generation," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 2130–2141. [Online]. Available: https://doi.org/10.1145/3510003.3510141

[4] E. Daka and G. Fraser, "A survey on unit testing practices and problems," in *25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014, Naples, Italy, November 3-6, 2014*. IEEE Computer Society, 2014, pp. 201–211. [Online]. Available: https://doi.org/10.1109/ISSRE.2014.11

[5] C. Pacheco and M. D. Ernst, "Randoop: feedback-directed random testing for java," in *Companion to the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada*. ACM, 2007, pp. 815–816. [Online]. Available: https://doi.org/10.1145/1297846.1297902

[6] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*. ACM, 2011, pp. 416–419. [Online]. Available: https://doi.org/10.1145/2025113.2025179

[7] M. M. Almasi, H. Hemmati, G. Fraser, A. Arcuri, and J. Benefelds, "An industrial evaluation of unit test generation: Finding real faults in a financial application," in *39th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, Buenos Aires, Argentina, May 20-28, 2017*. IEEE Computer Society, 2017, pp. 263–272. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2017.27

[8] C. Watson, M. Tufano, K. Moran, G. Bavota, and D. Poshyvanyk, "On learning meaningful assert statements for unit test cases," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1398–1409. [Online]. Available: https://doi.org/10.1145/3377811.3380429

[9] H. Yu, Y. Lou, K. Sun, D. Ran, T. Xie, D. Hao, Y. Li, G. Li, and Q. Wang, "Automated assertion generation via information retrieval and its integration with deep learning," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 163–174. [Online]. Available: https://doi.org/10.1145/3510003.3510149

[10] T. Tanimoto, *An Elementary Mathematical Theory of Classification and Prediction*. International Business Machines Corporation, 1958. [Online]. Available: https://books.google.com.hk/books?id=yp34HAAACAAJ

[11] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep api learning," in *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*, 2016, pp. 631–642.

[12] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 933–944.

[13] Z. Chen, S. Kommrusch, M. Tufano, L. Pouchet, D. Poshyvanyk, and M. Monperrus, "Sequencer: Sequence-to-sequence learning for end-to-end program repair," *IEEE Trans. Software Eng.*, vol. 47, no. 9, pp. 1943–1959, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2940179

[14] H. Hata, E. Shihab, and G. Neubig, "Learning to generate corrective patches using neural machine translation," *CoRR*, vol. abs/1812.07170, 2018. [Online]. Available: http://arxiv.org/abs/1812.07170

[15] A. Mesbah, A. Rice, E. Johnston, N. Glorioso, and E. Aftandilian, "Deepdelta: learning to repair compilation errors," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. ACM, 2019, pp. 925–936. [Online]. Available: https://doi.org/10.1145/3338906.3340455

[16] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, "An empirical investigation into learning bug-fixing patches in the wild via neural machine translation," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. ACM, 2018, pp. 832–837. [Online]. Available: https://doi.org/10.1145/3238147.3240732

[17] Y. Lou, Q. Zhu, J. Dong, X. Li, Z. Sun, D. Hao, L. Zhang, and L. Zhang, "Boosting coverage-based fault localization via graph-based representation learning," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 664–676. [Online]. Available: https://doi.org/10.1145/3468264.3468580

[18] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*. ACM, 2018, pp. 200–210. [Online]. Available: https://doi.org/10.1145/3196321.3196334

[19] B. Li, M. Yan, X. Xia, X. Hu, G. Li, and D. Lo, "Deepcommenter: a deep code comment generation tool with hybrid lexical and syntactical information," in *ESEC/FSE'20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. ACM, 2020, pp. 1571–1575. [Online]. Available: https://doi.org/10.1145/3368089.3417926

[20] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, "Summarizing source code with transferred API knowledge," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. ijcai.org, 2018, pp. 2269–2275. [Online]. Available: https://doi.org/10.24963/ijcai.2018/314

[21] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, "Retrieval-based neural source code summarization," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1385–1397.

[22] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*. IEEE, 2020, pp. 261–271. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054857

[23] H. Wei and M. Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. ijcai.org, 2017, pp. 3034–3040. [Online]. Available: https://doi.org/10.24963/ijcai.2017/423

[24] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," in *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019*, Y. Gu´eh´eneuc, F. Khomh, and F. Sarro, Eds. IEEE / ACM, 2019, pp. 70–80. [Online]. Available: https://doi.org/10.1109/ICPC.2019.00021

[25] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 2019, pp. 783–794. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00086

[26] G. Zhao and J. Huang, "Deepsim: deep learning code functional similarity," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. ACM, 2018, pp. 141–151. [Online]. Available: https://doi.org/10.1145/3236024.3236068

[27] M. Tufano, D. Drain, A. Svyatkovskiy, and N. Sundaresan, "Generating accurate assert statements for unit test cases using pretrained transformers," in *IEEE/ACM International Conference on Automation of Software Test, AST@ICSE 2022, Pittsburgh, PA, USA, May 21-22, 2022*. ACM/IEEE, 2022, pp. 54–64. [Online]. Available: https://doi.org/10.1145/3524481.3527220

[28] A. Mastropaolo, S. Scalabrino, N. Cooper, D. Nader-Palacio, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Studying the usage of text-to-text transfer transformer to support code-related tasks," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 336–347. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00041

[29] A. Mastropaolo, N. Cooper, D. Nader-Palacio, S. Scalabrino, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Using transfer learning for code-related tasks," *IEEE Trans. Software Eng.*, vol. 49, no. 4, pp. 1580–1598, 2023. [Online]. Available: https: //doi.org/10.1109/TSE.2022.3183297

[30] "ToGA Artifact." https://github.com/microsoft/toga, 2022.

[31] "A issue bug of ToGA Artifact." https://github.com/microsoft/toga/issues/3, 2022.

[32] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. ACM, 2018, pp. 373–384. [Online]. Available: https://doi.org/10.1145/3238147.3238190

[33] "*INT* Artifact." https://github.com/yh1105/Artifact-of-Assertion-ICSE22, 2022.

[34] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=SyyGPP0TZ

[35] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[36] C. Thunes, "Javalang." https://github.com/c2nes/javalang, 2019.

[37] J. Li, Y. Li, G. Li, X. Hu, X. Xia, and Z. Jin, "Editsum: A retrieve-and-edit framework for source code summarization," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 155–166. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678724

[38] Z. Liu, X. Xia, M. Yan, and S. Li, "Automating just-in-time comment updating," in *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 2020, pp. 585–597. [Online]. Available: https://doi.org/10.1145/3324884.3416581

[39] Z. Liu, X. Xia, D. Lo, M. Yan, and S. Li, "Just-in-time obsolete comment detection and update," *IEEE Trans. Software Eng.*, vol. 49, no. 1, pp. 1–23, 2023. [Online]. Available: https://doi.org/10.1109/TSE.2021.3138909

[40] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*. European Language Resources Association (ELRA), 2018. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2018/summaries/627.html

[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[42] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. The Association for Computational Linguistics, 2015, pp. 1412–1421. [Online]. Available: https://doi.org/10.18653/v1/d15-1166

[43] L. R. Dice, "Measures of the amount of ecologic association between species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[44] W. contributors., "Overlap—Wikipedia." https://en. wikipedia. org/w/index. php?title=Overlap&oldid=1061948530, 2021.

[45] F. Hassan and X. Wang, "Hirebuild: an automatic approach to history-driven repair of build scripts," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 1078–1089. [Online]. Available: https://doi.org/10.1145/3180155.3180181

[46] Y. Lou, Z. Chen, Y. Cao, D. Hao, and L. Zhang, "Understanding build issue resolution in practice: symptoms and fix patterns," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 617–628. [Online]. Available: https://doi.org/10.1145/3368089.3409760