


# 完美是测试预言机的敌人

Ali Reza Ibrahimzada 伊利诺伊大学厄巴纳-香槟分校，美国 alirezai@illinois.edu

Yigit Varli 中东技术大学，土耳其 yigit.varli@metu.edu.tr

Dilara Tekinoglu 马萨诸塞大学阿默斯特分校，美国 dtekinoglu@umass.edu

Reyhaneh Jabbarvand 伊利诺伊大学厄巴纳-香槟分校，美国 reyhaneh@illinois.edu

## 摘要

测试预言机的自动化是软件测试中最具挑战性的方面之一，但与自动化测试输入生成相比，受到的关注仍然相对较少。测试预言机依赖真实标签来区分正确行为与有缺陷的行为，从而确定测试是失败（检测到缺陷）还是通过。预言机问题之所以具有挑战性且不可判定，是因为它假设真实标签必须知道精确的预期行为、正确行为或有缺陷的行为。然而，我们认为，即使不知道精确的正确行为或有缺陷的行为，只要了解二者可能存在的差异，仍然可以构建准确的预言机。本文介绍了 SEER，这是一种基于学习的方法，在没有测试断言或其他类型预言机的情况下，可以判断单元测试在给定被测方法（MUT）上的结果是通过还是失败。为构建真实标签，SEER 将单元测试与被测方法的实现共同嵌入统一向量空间，使测试的神经表示与其能够通过的被测方法相似，而与其无法通过的被测方法不同。基于这种向量表示构建的分类器充当预言机：当测试输入检测到 MUT 中存在缺陷时生成“失败”标签，否则生成“通过”标签。我们在来自多种开源 Java 项目的 5K 多个单元测试上开展了大量实验，结果表明，所生成的预言机（1）在预测失败或通过标签方面效果出色，准确率、精确率、召回率和 F1 值分别达到 93%、86%、94% 和 90%；（2）具有良好的泛化能力，对于训练集或验证集中未包含的项目的单元测试，也能预测其标签，且性能下降可以忽略不计；（3）效率高，平均仅用 6.5 毫秒即可检测缺陷是否存在。此外，通过解读神经模型，并将其视为一个非封闭的解决方案，我们确认该预言机是有效的，即它通过学习相关特征来预测标签。

## 关键词

软件测试、测试预言机、测试自动化、深度学习

**ACM 参考文献格式：**

Ali Reza Ibrahimzada、Yigit Varli、Dilara Tekinoglu 和 Reyhaneh Jab- barvand。2022。完美是测试预言机的敌人。载于第 *30* 届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议论文集（*ESEC/FSE '22*），*2022*年*11*月*14*日至*18*日，新加坡，新加坡。ACM，纽约，纽约州，美国，12 页。https://doi.org/10.1145/3540250.3549086

## 1 引言

与图 1中的示例类似，单元测试由四个主要组件组成：测试输入（"e1"、"e2" 和 "e3"）、被测方法调用（`obj.sort()`）、测试输出和测试预言机（`assertEquals`）。给定一个了解程序针对给定输入的预期正确行为或有缺陷行为的真实标签，预言机便可判定测试结果，即测试是通过还是失败。例如，图1 中的真实标签将给定输入对应的排序输出定义为 "e1,e2,e3"。因此，断言预言机会检查生成的输出是否与预期结果一致，从而得出测试结果。构建真实标签可能是一项艰巨的任务。此外，缺少自动化真实标签意味着需要人工判断生成的输出是否正确，这构成了一个显著瓶颈，阻碍了全面实现测试自动化 [6]。

```
@Test
public void 测试Add() {Example对象 对象 =
                        对象.add("e1");对象.add("e2");
                        ing output = 对象.
                        tEquals(output,"
```

**图 1：一个简单的 JUnit 测试由四个主要组件组成：输入、被测方法调用、输出和断言**

为了自动构建测试预言机的真实标签，传统技术和基于机器学习的技术依赖现有或推导出的形式化规格说明，断言 [18], 程序不变量 [25, 51, 72, 77, 80],以及蜕变关系 [23, 88], 来识别正确行为。其他一些技术则会确定测试执行期间观察到的特定类型缺陷所对应的模式，并将其作为有缺陷行为的指标 [13, 14, 87, 90]。 [23, 88], 这些技术的共同之处在于，它们都着重识别精确的正确行为或有缺陷的行为，以构建真实标签。然而，识别精确的行为和输出是一个

## CCS 概念

• **软件及其工程** → **软件测试与调试ging**；• **计算方法** → **神经网络**。

允许为个人或课堂使用免费制作本作品全部或部分内容的数字或纸质副本，但不得为营利或商业利益制作或分发副本，且副本必须附上此声明及首页完整引文。对于本作品中归他人所有的组件，必须尊重其版权。允许在注明出处的情况下摘录。若要以其他方式复制、再版、发布到服务器或分发至列表，须事先获得特定许可和/或支付费用。请通过 permissions@acm.org 联系我们申请许可。*ESEC/FSE '22*，*2022*年*11*月*14*日至*18*日，新加坡，新加坡©2022 计算机协会。ACM ISBN 978-1-4503-9413-0/22/11...$15.00https://doi.org/10.1145/3540250.3549086

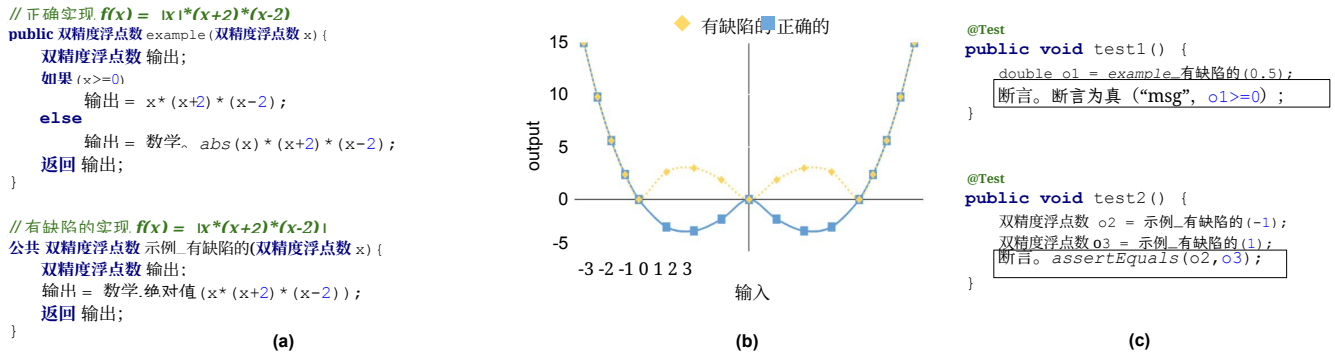



**图 2：(a) 数学函数 $f(x) = |x|\times(x+2)\times(x-2)$ 正确（上）和有缺陷的（下）实现，(b) 正确和有缺陷实现行为的可视化，(c) 用于评估有缺陷实现正确性的、包含断言的 JUnit 测试**

不可判定问题；因此，这类技术只能对程序进行部分验证。也就是说，只有在已知预期输出的部分测试输入下，才能验证程序；或者只能验证由不变量或蜕变关系确定的程序有限属性。尽管部分预言机在一定程度上提升了测试自动化水平，但它们未必能保证存在成功的测试驱动程序。

我们研究中的关键洞见是，即使不知道正确或有缺陷行为下输入与输出之间的明确关系，仍然可以创建准确的测试预言机。相反，真实标签将决定测试输入与正确和有缺陷行为下的输出之间的相关程度。基于此真实标签的预言机无需使用断言语句，也无需在断言中确定确切的预期输出，从而极大地提升单元测试 [77]。

本文介绍了 SEER [1]，这是一种用于预测单元测试结果的自动化预言机。具体而言，给定一对 $<t_i, m_i>$，其中 $t_i$表示一个没有任何断言的单元测试，而 $m_i$ 表示 MUT 的实现，SEER 会自动判断该测试在 MUT 上是通过还是失败。为构建真实标签，SEER 利用联合嵌入来区分正确和有缺陷的 MUT 的神经表示。基于该嵌入构建的分类器能够学习输入与输出之间的相关性，从而预测测试结果。本文的贡献如下：

- 一种面向单元测试和 MUT 的新型领域特定联合嵌入，可根据单元测试在 MUT 上通过还是失败，在语义上区分 MUT 的神经表示。

- 设计了一种可解释的深度学习模型，作为测试预言机，为没有断言的单元测试生成通过或失败标签。这种可解释性使我们能够超越将深度学习作为黑盒技术的用法，并通过查看 MUT 实现中的相关词元，验证模型是否据此预测标签。虽然这不属于本文的讨论范围，但模型决策所涉及的相关词元还可供开发人员进一步用于定位检测到的缺陷。

- 针对广泛使用的开源 Java 程序进行的大规模实证评估表明，SEER（1）有效——总体准确率、精确率、召回率和 F1 度量分别达到 93%、86%、94% 和 90%；（2）具有泛化能力——能够预测不在训练集或验证集中的项目的单元测试标签，且性能几乎没有下降；（3）高效——训练完成后，平均仅需 6.5 毫秒即可检测出缺陷是否存在。SEER的实现和相关资料已公开 [2]。

本文其余部分安排如下。第2 节展示一个说明性示例。第3 节概述SEER，第4 节介绍所提技术的详细内容。第5 节展示评估结果。最后，本文讨论相关研究与未来工作。

## 2 说明性示例

为说明先前工作的局限性并阐释我们研究背后的直觉，我们采用图2-a 所示的两个代码示例。上方的代码片段是一个数学函数的正确的实现，它计算的输出为

$$|x|\times(x+2)\times(x-2)。$$

而有缺陷的版本计算的输出为 $|x\times(x+2)\times(x-2)|$ 这是因为一个括号被移到了赋值语句末尾，而非放在变量x之后。

形式上，代码的行为是一个函数 $B: I \rightarrow O$ 它将 $I$中的输入映射到 $O$中相应的输出。在我们的示例中，表示正确实现和有缺陷实现显式行为的映射函数，分别以蓝色和黄色曲线绘于图2-b 中。如果已知这样的函数，预言机就可以将其作为真实标签，以区分有缺陷的行为和正确的行为。然而，实际上，被测方法会接收多个复杂输入，例如数组和用户定义的对象，从而形成输入与输出之间难以确定的 n 维映射。因此，测试预言机依赖部分真实值。无论我们如何构建真实标签，对于属于$(-2, 0)\cup(0, 2)$的测试输入，预言机都应判定为“失败”，因为在此区间内，正确实现与有缺陷实现的行为有所不同。

假设我们有图中所示的两个 JUnit 测试 2-c，用于评估示例_有缺陷的方法的正确性。test1中断言语句的预期输出的基准真值基于动态不变量检测，而test2 中断言语句的基准真值则基于蜕变关系 $f(x) = f(-x)$。动态不变量检测技术依赖于现有代码的执行轨迹。由于我们的 MUT 有缺陷，该不变量只捕获有缺陷行为的属性，即 $output \geq 0$。通过检查断言中生成的不变量，

[1]能够凭借超自然洞察力预见未来的人。

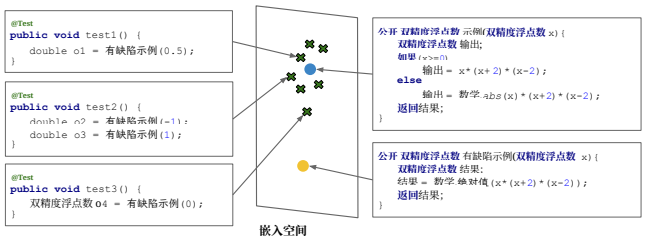



**图 3：测试与被测方法联合嵌入背后的直觉，旨在根据测试区分有缺陷和正确的被测方法的表示**

测试 通过，对于 $x = 0.5 \in (-2, 0) \cup (0, 2)$，但它本应失败，以展示这个缺陷。同样，$f(x) = f(-x)$ 的蜕变关系对正确和有缺陷的实现都成立；因此，该断言错误地将非重叠范围内的测试输入判定为通过。

此示例表明，明确识别正确（或有缺陷的）行为以构建真实标签——这是以往研究的重点——存在显著局限。在本研究中，我们不再试图了解在正确或有缺陷的行为下，输入与输出之间明确的关联，而是旨在学习：对于失败和通过的配对，输入与输出之间有何种不同的相关性 $<t_i, m_i>$。此处，$t_i$ 表示不含任何断言的单元测试，而 $m_i$ 表示 MUT 的实现。为此，SEER 学习 MUT 和测试的向量表示，使测试的向量表示与其能够通过的 MUT 相似，而与其能够揭示缺陷、即测试会失败的 MUT 不相似。这种联合嵌入在 n 维向量空间中区分了有缺陷的 MUT 和正确的 MUT。

图 3 展示了SEER中联合嵌入背后的直观原理。由于test1和 test2 中的测试输入能够揭示示例_有缺陷的MUT 中的缺陷，因此它们在嵌入空间中更靠近正确的 MUT，而离有缺陷的 MUT 更远；也就是说，它们与正确 MUT 的向量表示相似，而与有缺陷 MUT 的向量表示不同。另一方面，test3 无法揭示缺陷，并且应当在有缺陷和正确的 MUT 上都通过，因此它在嵌入空间中与正确和有缺陷的 MUT 距离相同。与图2-b 相比，测试输入与正确或有缺陷行为下的输出之间没有明确关系；但正确和有缺陷 MUT 的嵌入表示有所区别，这是基于它们生成的输出与通过或失败测试的输入之间存在不同的相关性。

## 3 框架概述

图 4 概述了由四个主要组件构成的 SEER 框架：(1) 方法提取器、(2) 数据集增强器、(3) 学习模块和 (4) 解释器。SEER需要高质量且规模庞大的 $<t_i, m_i>$ 实例数据集来训练预言机。给定一组程序及其对应的测试套件，方法提取器组件通过轻量级静态分析，从测试中提取被测方法调用的实现，构建这样的数据集（详情见 §4.1）。接下来，方法提取器会创建带标签的元组，形式为 $\langle t_i, m_i, l_i \rangle$，其中 $t_i$ 表示一个单元测试，$m_i$ 表示被测方法的实现，而 $l_i$ 表示测试结果，即可能是 $P$ (通过)或 $F$ (失败)。数据集增强器组件随后会对生成的数据集进行增强，添加更多实例以丰富缺陷类型并处理不平衡标签（详情见 §4.3）。

训练数据集准备就绪后，SEER 将其输入学习模块，通过两个阶段训练预言机。在阶段1训练中，学习模块通过联合嵌入学习测试 $t_i$ 和 MUT $m_i$ 的向量表示，最小化通过元组 '$t_i, m_i, P$( 之间的距离，同时最大化失败元组 '$t_i, m_i, F$( 之间的距离（详见§4.2）。因此，测试的向量表示与其通过的 MUT 相似，但与其失败的 MUT 不相似。在学习到具有判别性的向量表示后，学习模块利用迁移学习 [52]，并在嵌入网络之上训练分类器，将其作为我们的测试预言机。为了预测标签，SEER 将单元测试和被测程序作为输入，并提取被调用 MUT(s) 的实现。给定生成的 '$t_i, m_i$( 对，嵌入网络首先计算它们的向量表示，然后由分类器预测标签，以表明测试在给定 MUT 上是通过还是失败。

SEER 不仅将深度学习用作黑盒方法，还会对学习到的模型进行解释，以实现两个目的：(1) 验证嵌入是否发挥了作用，即根据被测方法是否通过或未通过测试，将其表示区分开来；(2) 通过检查影响预言机决策的代码词元是否相关，验证模型的有效性（§4.4）。下一节将介绍 SEER各组件的详情。

## 4 SEER

本节首先说明如何为学习模块准备输入，随后介绍 SEER 的神经网络架构、数据集整理和模型解释的详情。

### 4.1 方法提取器

SEER 的学习模块需要带标签的配对 $<t_i, m_i>$，以实现输入（由单元测试 $t_i$ 提供）与输出（由 MUT $m_i$ 生成）之间的关联。给定一个由 $n$ 个单元测试组成的测试套件 $T = \{t_1, t_2, \ldots, t_n\}$，以及一个由 $k$ 个开发者编写的方法[2]组成的被测程序 $P = \{m_1, m_2, \ldots, m_k\}$，方法提取器会提取 $m_i$，即在单元测试 $t_i$ 的方法体中被直接调用的 MUT 的实现。

方法提取可以静态执行，即提取被测方法的整个方法体，而不考虑给定测试覆盖的语句；也可以通过动态分析执行，即只考虑测试已执行的语句。虽然后一种方式更直观，有助于模型聚焦于已执行的代码行来预测测试判定，但近期研究表明，提供全局信息时，神经模型能够更有效地学习代码及上下文的语义 效果更佳 [34, 76]。因此，模型提取器 会对给定的单元测试执行轻量级 流敏感分析 $t_i$，识别属于被测程序的 方法调用，并提取相应的方法签名和方法体。如果测试调用了多个方法，模型提取器 会按照调用顺序，串联被测方法的提取信息。在图2的说明性示例中，方法提取器识别出

[2]我们排除第三方 API，因为可能无法获取其代码。

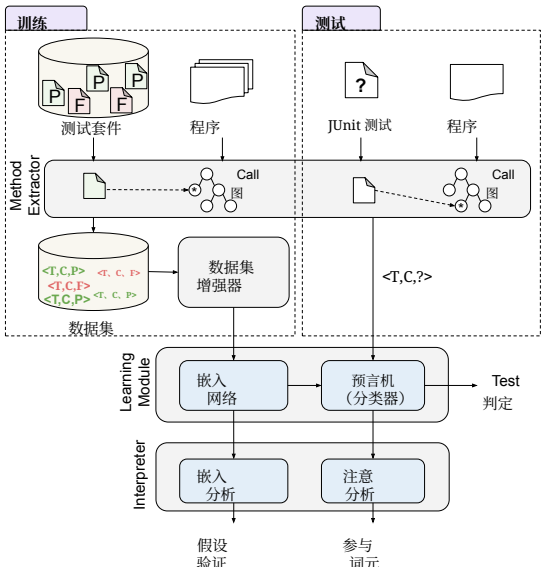



**图 4：SEER 框架概览**

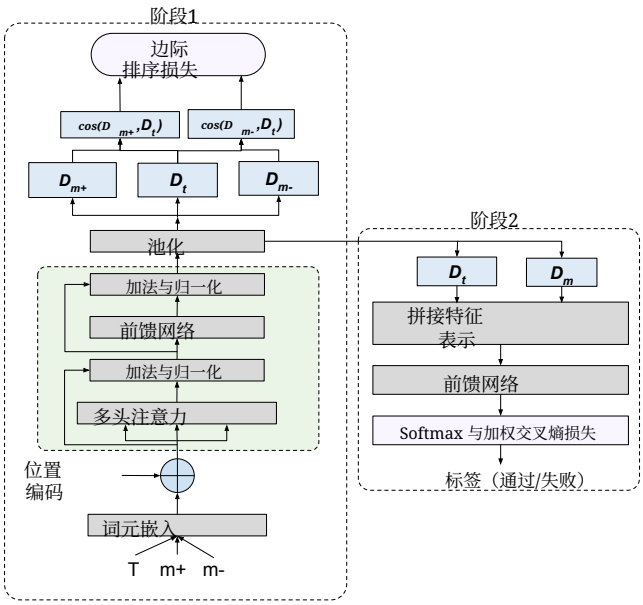



**图 5：SEER 的总体架构**

示例_有缺陷的方法 $m_i$，并提取该方法的全部文本，包括方法签名和方法体。

## 4.2 学习模块

SEER 的神经网络架构SEER 如图 5所示。充当预言机的神经模型分两个阶段进行学习。在第一阶段训练中，SEER 学习单元测试和被测方法的向量表示，使有缺陷的和正确的被测方法的表示有所不同。在下一步中，以下表示<$t_i$，$m_i$>对将输入分类器，帮助 SEER学习由以下各方提供的输入 $t_i$与由以下各方产生的输出之间的相关性$m_i$。由于有缺陷的和正确的被测方法所产生的表示有所不同，预言机最终会学习在正确和有缺陷的行为下，输入与输出之间的相关性有何不同。

在第一阶段训练期间，SEER 通过联合嵌入 [84]学习单元测试和被测方法的向量表征。联合嵌入也称为多模态嵌入，已广泛用于将异构数据嵌入统一向量空间，使两种模态中语义相似的概念在嵌入空间中彼此更接近。例如，在计算机视觉领域，研究人员使用卷积神经网络（CNN）和循环神经网络（RNN），将图像和文本联合嵌入同一向量空间，以便为图像添加标签 [39]。

在我们的问题中，我们采用联合嵌入的概念，根据测试结果在语义上区分正确和有缺陷的 MUT 的表示。具体而言，我们假设，通过将单元测试和 MUT 联合嵌入到统一向量空间中，使测试与其通过的 MUT 具有相似的向量表示，而与其未能通过的 MUT 具有不同的向量表示，由此得到的嵌入能够区分正确和有缺陷的 MUT 的表示。单元测试 $t_i$ 与被测方法的实现 $m_i$ 的联合嵌入可表示如下：

$$m_i \xrightarrow{\phi} D_{m_i} \rightarrow J(D_{m_i}, D_{t_i}) \leftarrow D_{t_i} \xleftarrow{\psi} t_i$$

其中，$\phi$ 是一个嵌入函数，用于将 $m_i$ 映射到一个 d 维向量空间 $D$；$\psi$ 是一个嵌入函数，用于将 $t_i$ 映射到同一向量空间 $D$；而 $J$，是一种相似度度量，用于评估 $D_{m_i}$ 与 $D_{t_i}$ 的匹配程度，以便在嵌入空间中使 $m_i$ 与 $t_i$ 更接近或更远。SEER 使用余弦相似度度量来衡量 $t_i$ 与 $m_i$ 的向量表示之间的相似度。较小的余弦相似度意味着两个向量在 d 维嵌入空间中彼此更接近，而较大的余弦相似度意味着向量指向不同的角度，即在嵌入空间中彼此相距更远。为学习并在语义上区分与测试结果相关的正确和有缺陷的 $m_i$，SEER 最小化如下排序损失：

$$\mathcal{L}(\theta) = \sum_{\langle t_i, m_{i+}, m_{i-} \rangle} max(cos(D_{t_i} - D_{m_{i+}}) - cos(D_{t_i} - D_{m_{i-}}) + \alpha, 0)$$

其中，$D_{t_i}$ 是 $t_i$的向量表征，$D_{m_{i+}}$ 是 $t_i$ 通过的被测方法的向量表征，$D_{m_{i-}}$ 是 $t_i$ 失败的被测方法的向量表征。$\theta$和 $\alpha$ 分别表示模型参数和常数间隔值。直观而言，通过最小化边际排序损失，SEER 学会缩小 $D_{t_i}$ 与 $D_{m_{i+}}$之间的距离，同时增大 $D_{t_i}$ 与 $D_{m_{i-}}$之间的距离。

在阶段1学习测试、 $D_{t_i}$和被测方法、$D_{m_i}$的向量表示后，SEER 将它们拼接起来，为 <$t_i$，$m_i$>对创建一个连续的特征表示。在第二阶段训练中，生成的组合特征向量会被输入一系列全连接层，以将学习到的特征解码为特定的目标类别，即通过或失败。

## 4.3 数据集整理

SEER 的训练需要一个大规模且高质量的数据集，即由通过和失败的'$t_i$，$c_i$( 对组成的数据集，这些样例代表了不同项目中的多种缺陷。为了构建该

数据集，我们从 Defects4J [1], 入手。Defects4J 是一个收录大型且广泛使用的 Java 项目中可复现缺陷的集合。该数据集中的每个缺陷都附带研究对象程序的有缺陷版本和修复版本，以及开发者编写的通过和失败测试。我们基于 Defects4J 构建数据集的理由是：（1）缺陷彼此独立且可复现，这使神经模型更容易学习相关特征；（2）它包含开发者编写的失败测试，有助于生成均衡的数据集，因为使用 Randoop [51] 和 EvoSuite [25] 自动生成能够触发失败的测试并无保证。

Defects4J 数据集的一个重要局限在于缺陷的复杂度较低，即大多数缺陷只涉及代码中的一条语句。这一问题可能通过两种方式降低 SEER 的性能。首先，模型可能会将代码中的微小改动视为噪声，并在学习时忽略这些改动，从而导致性能下降 [33, 79]。更重要的是，在简单缺陷上训练的模型可能无法泛化到复杂且更贴近实际的缺陷。受 mutation testing 在整理高质量训练数据集方面的强大作用 [35]，以及高阶变异体 [37] 更能代表复杂缺陷这一事实的启发，SEER 的数据集增强器 组件将一个通过的 $'t_i$, $m_i$( 对作为输入，并在不同位置反复对 MUT 进行变异，以生成高阶 变异体。

算法1 说明了我们的数据集增强流程。该算法以 *MUT* 和 *order*（对给定 MUT 进行变异的最大次数）为输入，并生成一个高阶变异体，*HOM*。为此，它首先识别出以下内容的唯一配对 $Mutables = \langle op,\ loc \rangle$ 这些配对展示了 *loc* 在 MUT 中可以应用变异算子的位置 *op*（第 1 行）。接下来，它使用这些配对，每次对 MUT 进行一次变异（第 4–5 行）。每次变异后，算法都会检查生成的变异体是否可编译（第 7 行）。如果变异体可编译，算法便使用下一个 *mutable*(继续变异（第 8–9 行）。否则，它会撤销此次变异，并以生成的 *HOM*(结束（第 10–11 行）。变异会持续进行，直到 MUT 已变异 *order*次，或所有可变异位置均已完成变异。

**算法1：** 数据集增强算法

```
输入：MUT，阶数
输出：MUT 的高阶变异体（HOM）
1  可变项 ← getUniqueMutables (MUT)
2  计数器 ←顺序
3  HOM ←MUT
4  对每个可变项 ∈ 可变项 执行
5      突变体 ← 突变 (MUT, 可变项.op, 可变项.loc)
6      如果 计数器 > 0那么
7          如果 isCompilable(突变体)那么
8              HOM ←变异体
9              计数器 ← 计数器 − 1
10         else
11             返回 HOM
12     else
13         跳出
14 返回 HOM
```

## 4.4 解释

没有解释，就无法信任机器学习（ML），尤其是深度学习（DL）模型的性能，因为它们的学习依赖数百万个参数。具体而言，这类智能模型可能产生不切实际的良好预测，但其依据却是从无关特征中学习得出的，这可能源于数据集中的噪声或数据泄漏问题 [40]。为确保 SEER 的可信性，我们通过解释学习模块来验证以下两个假设：假设*1*。预言机关注 MUT 中的相关词元，以预测测试结果；以及 (2) 假设*2*。嵌入网络通过区分被测方法的向量表示，在嵌入空间中分离有缺陷的和正确的 MUT。我们将分别讨论注意力分析和嵌入分析的详细信息，以研究假设*1*和假设*2*的正确性。

*4.4.1* 注意力分析。 注意力机制 [15],最初是为了解决循环神经网络（RNN）中的长序列问题 而提出的，它是一种帮助深度学习模型在执行任务时识别特征序列中各个特征重要性的方法。注意力机制在神经网络架构中有两个作用；首先，它有助于提升模型的性能。

更重要的是，它已被广泛用于解决深度神经模型的可解释性问题。注意力机制最初的实现形式是神经网络架构中编码器与解码器组件之间的神经网络层，并生成注意力权重向量 $\overrightarrow{AT} = \{w_0, \ldots\ldots, w_n\}$作为输出。在神经代码分析的上下文中，$w_i$ 表示：给定一条包含 $n$ 个词元的语句，在预测标签时，语句中位置 $i$ 处的词元有多重要。某个特征的注意力权重越高，模型在进行预测时就越关注该特征。

SEER 使用多头注意力 [15, 74]，（也称为自注意力），在学习时考虑语句中词元的相对重要性。自注意力层的输出是一个 $n \times n$矩阵 $SA = [[w_{0_0}, \ldots, w_{0_n}], \ldots, [w_{n_0}, \ldots, w_{n_n}]]$，其中$w_{i_j}$ 表示位置 $i$ 处给定特定词元的重要程度 $j$。图6 展示了这两种注意力机制在我们的有缺陷的语句中的差异

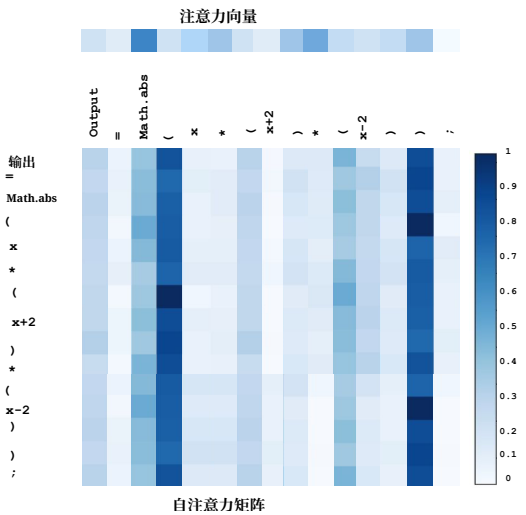



**图6：自注意力在预言机问题中的重要性**

**算法2：** 注意力分析

**输入：** MUT 的词元 $\overrightarrow{Tkn} = \{c_0, \ldots\ldots, c_n\}$
**输入：** MUT 的语句 $\overrightarrow{Smt} = \{s_0, \ldots\ldots, s_m\}$
**输入：** $SA = [[w_{0_0}, \ldots, w_{0_n}], \ldots, [w_{n_0}, \ldots, w_{n_n}]]$
**输入：** 注意力阈值 $k$
**输出：** 受关注词元 $ATkn$，受关注语句 $ASmt$

1 $ATkn \leftarrow \emptyset$
2 $ASmt \leftarrow \emptyset$
3 **遍历** $row = [w_{i_0}, \ldots, w_{i_n}] \in SA$，**执行**
  // $localATkn = \{' c_i,\ ind_i (|\ ind_i$ 是 $c_i$ 在 $\overrightarrow{Tkn}$ 中的索引}
4  $localATkn \leftarrow getMostAttended(row, Tkn, k)$
5  **对每个** $\langle c_i, ind_i$ （ $\in localATkn$ **执行**
6  **如果** $\neg ATkn.contains(\langle c_i, ind_i$ （） **则**
7  $ATkn \leftarrow ATkn \cup c_i$
8 **对每个** $s_i \in Smt$ **执行**
9  **如果** $|s_i \cap ATkn| > k$ **则**
10  $ASmt \leftarrow ASmt \cup s_i$

说明性示例（为了节省篇幅并提高可读性，图6 仅展示了与错误语句对应的部分 SA）。（图2-a）。如该图所示，与传统注意力机制相比，自注意力在捕捉右括号相对于左括号的重要性方面更加成功。算法2介绍了SEER的注意力分析方法，即分析 $SA$ 矩阵，以确定在 MUT 中受到最多关注的词元和语句，从而预测测试结果。

对于给定的 '$t_i$,$m_i$ 对，算法2 将 MUT 的词元 {$c_0$、……、$c_n$}、自注意力矩阵 $SA = [[w_{0_0}, \ldots, w_{0_n}], \ldots, [w_{n_0}, \ldots, w_{n_n}]]$ 和注意力阈值 $k$ 作为输入，以识别受关注词元集合 $ATkn$ 和受关注语句 $ASmt$，并将其作为输出。为此，该算法逐行遍历 $SA$ 矩阵（第 3—7 行），识别受关注程度最高的前 $k$%个词元——即注意力权重值最高的前 $k$ 个词元（第 4 行），并将整个矩阵中每一行的受关注词元及其对应的索引合并起来（第 5—7 行）。合并的结果就是受关注词元集合 $ATkn$。在合并时，考虑索引尤为重要，因为位于不同索引处的相同词元可能会受到不同程度的关注。在图6的示例中，尽管词元"("在 SA 的不同索引处多次出现，但其最高注意力位于最后一个索引处。

最后，算法遍历 MUT 的语句 {$s_0$、……、$s_m$}，并确定其中词元与 $ATkn$ 中的受关注词元重叠比例达到 $k$% 的语句（第 8—10 行）。当某个 '$t_i$,$m_i$( 对的预测标签为 " 失败 " 时，这些语句表明了有缺陷的语句。这里的直觉是，由于缺陷代码行的数量有限，根据广义鸽巢原理 [10]，，受关注词元中至少有一个语句包含超过 $k$% 的词元 $ATkn$。

*4.4.2* 嵌入分析。 SEER 依靠可视化技术来验证有缺陷的和正确的 MUT 表示在嵌入空间中的分离情况。然而，鉴于嵌入向量的高维性，嵌入分析的第一步是降低表示的维度。PCA [53], 线性判别分析 [70], 和 t-SNE [29]

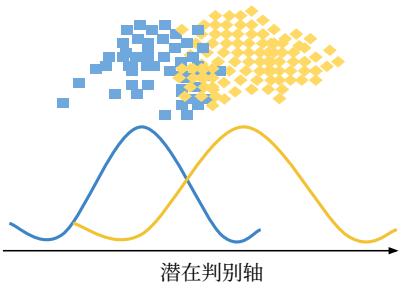



**图 7：线性判别分析降维背后的直觉**

专注于将不相似的数据点置于低维表示中的较远位置。

在最流行的降维算法中，SEER 采用线性判别分析（LDA），因为它能够识别类别标签，并在降维过程中最大化类别之间的分离度。图 7 展示了 LDA 如何进行高维降维的直观示意。散点图表示高维嵌入空间中有缺陷（黄色）和正确（蓝色）MUT 的分布，曲线则表示它们在低维空间中的分布。如图 7 所示，如果两个类别的降维后实例分布重叠，则它们在高维空间中并未被正确分离。否则，有缺陷类别和正确类别的实例会在高维嵌入空间中彼此分离。

## 5 评估

为评估 SEER 的有效性，我们探讨以下研究问题：

**研究问题1：** 有效性。所提出的技术在准确预测测试通过或失败标签方面有多有效？所提出的预言机能够检测哪类缺陷？哪些缺陷更难被预言机检测到？

**研究问题2：** 泛化。所提出的技术在多大程度上能够为未参与训练的 Java 项目预测测试标签？**研究问题3：** 解释。嵌入是否真的能够区分通过和失败的 MUT 表示 <测试、MUT> 对？哪些特征会影响预言机的决策？

**研究问题4：** 性能。所提出技术的性能特征是什么？

### 5.1 实验设置

为便于复现，我们将详细说明实验设置。此外，我们已在 GitHub [2] 上公开了 SEER 的所有工件。

*5.1.1* 数据集：表1展示了用于训练、验证和测试SEER的数据集属性详情。我们的数据集基于 Defects4J 构建 [1]，，其中包含835 个缺陷，涉及17 个广泛使用的 Java 项目。除了从 Defects4J 收集的数据外，我们还通过高阶变异体和自动生成的 Randoop [51] 测试扩充了数据集。前者有助于丰富数据集中的缺陷类型，后者则生成在新添加的缺陷上通过或失败的测试。最后，我们

**表 1：数据集的属性。**

| 项目 | # 缺陷 | # 变异体 | 数据集 Defects4J #测试 #通过，#失败（贡献度%） | 开发者测试 # 通过 | 开发者测试 # 失败 | Randoop 测试 # 通过 | Randoop 测试 # 失败 | 高阶变异体 开发者测试 # 通过 | 高阶变异体 开发者测试 # 失败 | 高阶变异体 Randoop 测试 # 通过 | 高阶变异体 Randoop 测试 # 失败 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 压缩 | 47 | 24,322 | 30,753 (30.26%) | 1,214 | 22 | 2,965 | 2230 | 385 | 265 | 192 | 23,480 |
| Lang | 64 | 12,824 | 15,130（14.89%） | 702 | 59 | 1,249 | 296 | 397 | 509 | 1,124 | 10,794 |
| 图表 | 26 | 4,313 | 14,901（14.66%） | 248 | 37 | 5,316 | 4987 | 154 | 109 | 229 | 3,821 |
| Math | 106 | 9,488 | 13,580（13.36%） | 1,685 | 91 | 1,584 | 732 | 732 | 814 | 546 | 7,396 |
| 编解码器 | 18 | 8,973 | 10,210 (10.05%) | 271 | 11 | 648 | 307 | 375 | 54 | 93 | 8,451 |
| 闭包 | 174 | 1,615 | 3,170 (3.12%) | 1,431 | 8 | 113 | 3 | 768 | 76 | 49 | 722 |
| Jackson 数据绑定 | 112 | 581 | 2,818 (2.77%) | 2,133 | 27 | 77 | 0 | 410 | 94 | 42 | 35 |
| Time | 26 | 203 | 2,415 (2.38%) | 1,765 | 39 | 255 | 153 | 84 | 70 | 18 | 31 |
| Jsoup | 93 | 677 | 2,134 (2.1%) | 925 | 57 | 458 | 17 | 125 | 234 | 277 | 41 |
| Cli | 39 | 913 | 1,612 (1.59%) | 348 | 10 | 292 | 49 | 106 | 75 | 371 | 361 |
| Csv | 16 | 1,166 | 1,582 (1.56%) | 327 | 6 | 83 | 0 | 297 | 2 | 23 | 844 |
| Jackson 核心 | 26 | 941 | 1,420 (1.4%) | 331 | 15 | 105 | 28 | 320 | 163 | 136 | 322 |
| Gson | 18 | 531 | 1,386 (1.36%) | 678 | 15 | 93 | 69 | 295 | 92 | 144 | 0 |
| JxPath | 22 | 143 | 314（0.31%） | 93 | 1 | 74 | 3 | 0 | 1 | 105 | 37 |
| Mockito | 38 | 0 | 100（0.1%） | 56 | 39 | 5 | 0 | 0 | 0 | 0 | 0 |
| JacksonXml | 6 | 49 | 81（0.08%） | 19 | 1 | 9 | 3 | 28 | 12 | 2 | 7 |
| Collections | 4 | 0 | 7 (0.01%) | 4 | 1 | 2 | 0 | 0 | 0 | 0 | 0 |
| 总计 | 835 | 66,739 | 101,613 (100%) | 12,230 | 439 | 13,328 | 8,877 | 4,476 | 2,570 | 3,351 | 56,342 |

移除了开发者编写的测试和自动生成的测试中的所有断言语句，以避免从这些测试中学习时产生偏差。

如表 1 所示，最终的增强数据集由 33, 385 个通过的 '$t_i, m_i$( 配对和 68, 228 个失败的 '$t_i, m_i$( 配对组成，数据集中的配对总数为 101, 613 个 '$t_i, m_i$( 配对。在阶段1训练中，我们通过合并共同测试的通过和失败 '$t_i, m_i$( 配对，构造 '$t_i, m_i+, m_i-$( 元组。也就是说，对于给定的测试 $t_i$，我们找出所有 $m$通过配对和 $n$失败配对 '$t_i, m_i$(。如果 $m$和 $n$ 均非零，则可得到总计 $m \times n$ 个 '$t_i, m_i+, m_i-$( 元组。这为阶段 1 训练提供了 20, 759 个 '$t_i, m_i+, m_i-$( 元组，并将其划分为 90% 的训练实例、5% 的验证实例和 5% 的测试实例。对于阶段2训练，我们同样对原始数据集进行划分；该数据集包含 101, 613 个实例，如表 1 所示。

*5.1.2* 学习模块配置：我们使用 PyTorch [54] 开源库实现了 SEER 的 学习模块。多种因素会影响神经模型的学习过程和最终性能。对于决定算法从训练数据中学习效果的损失函数，我们在第一阶段训练中使用边际排序损失（MRL），在第二阶段训练中使用加权交叉熵损失（WCEL）。研究表明，在学习嵌入时，MRL 的表现优于交叉熵损失；它能使属于同一目标类别的数据实例彼此更接近，而与其他类别的实例保持距离 [30]。由于我们的嵌入目标与此类似，即使通过配对的 '$t_i$，$m_i$ (彼此更接近，而失败配对相对更远，因此 MRL 是学习 MUT 和测试嵌入的合理损失函数。

在第二阶段训练中，我们选择了加权交叉熵损失，而不是分类问题中常用的交叉熵损失，因为我们的数据集中"$t_i, m_i$("的失败配对比通过的配对更多。为了提升性能，我们采用了 AdamW 优化器 [44]，它已被证明优于 Adam 优化器 [41]，用于更新网络权重并迭代地最小化这一损失函数。

影响模型性能的其他因素是超参数和过拟合。我们采用有指导的超参数调优，为模型寻找能在验证数据上取得最佳性能的配置。最重要的超参数之一是学习率，它控制模型每次更新权重时根据估计误差调整模型的幅度。学习率的选择颇具挑战：数值过小可能导致训练过程漫长且陷入停滞，而数值过大则可能导致训练过程不稳定。SEER 的学习模块 在阶段1和第二阶段训练中的学习率分别为 $1.34e^{-4}$ 和 $1.34e^{-6}$。学习率不同，是因为与阶段1相比，阶段2的学习是增量式的；使用相近的学习率会导致损失函数值很大，即出现 NaN。此外，我们采用 10 折交叉验证来避免过拟合，并实施早停标准以终止训练。也就是说，我们在不同的训练集和验证集上重复进行训练/验证 10 次，并选择性能最佳的模型。为自动终止学习，我们将耐心值设为 5 个训练轮次；也就是说，如果模型的验证损失连续 5 个训练轮次没有改善，我们就认为学习已达到最优水平。

## 5.2 研究问题一：有效性

针对这一研究问题，我们将 101,613 组 $\langle T, C, F \rangle$数据集中的实例划分为 90% 训练集、5% 验证集和 5% 测试实例。为此，我们将每个项目中的此类实例下采样90%，并在可能的情况下使用剩余实例。唯一的例外是 Collections 项目，我们只将其中少量实例纳入训练集。我们选择准确率、精确率、召回率和 F1 分数作为指标，以衡量 ffectivenessof SEER预测正确标签的有效性。表 2 展示了该实验在"SEER with embedding"列下的结果。这些结果通过 10折交叉验证获得，即重复下采样10 次。

**表 2：SEER 在预测测试标签方面的有效性和泛化能力。TP、FP、TN 和 FN 分别代表真正例、假正例、真负例和假负例。**

| 受试对象 | 带嵌入的 SEER | |
|---|---|---|
| | # 通过 | # 失败 |
| | TP（%），FN（%） | TN（%），FP（%） |
| 压缩 | 92.53%，7.47% | 98.18%，1.82% |
| Lang | 81.07%，18.93% | 92.6%，7.4% |
| 图表 | 94.74%，5.26% | 88.03%，11.97% |
| Math | 93.83%，6.17% | 89.98%，10.02% |
| 编解码器 | 79.12%，20.88% | 99.58%，0.42% |
| 闭包 | 98.36%，1.64% | 80.56%，19.44% |
| Jackson 数据绑定* | 100%，0% | 30.77%，69.23% |
| Time* | 100%，0% | 0%，100% |
| Jsoup* | 98.81%，1.19% | 0%，100% |
| Cli | 96.23%，3.77% | 60%，40% |
| Csv | 94.59%，5.41% | 98.21%，1.79% |
| Jackson 核心 | 97.06%，2.94% | 41.94%，58.06% |
| Gson* | 100%，0% | 66.67%，33.33% |
| JxPath* | 100%，0% | 25%，75% |
| Mockito* | 100%，0% | 0%，100% |
| JacksonXml* | 100%，0% | 0%，100% |
| Collections* | 不适用，不适用 | 不适用，不适用 |
| 总计 | 93.63%，6.37% | 92.77%，7.23% |

表 2 中的每一行展示了一个研究对象项目，以及该项目对不同版本 ff的实例进行正确预测的百分比SEER。这些结果证fi实了图 **5 中展示的 SEER 原始实现能够有效预测每个研究对象程序的测试套件中的通过和失败标签，达到 93% 准确率、86%** 精确率、**94%** 召回率和 **90%** F1 分数。**尽管整体性能良好，**SEER **在一些项目上表现不佳**（**表** 2中以星号标记的项目）。我们的调查显示，由于这些项目对数据集的贡献度较低，其中的测试数据实例要么不存在（例如 Collections 项目），要么非常少。因此，分类的效应量非常大。

在我们提出的技术中，假阴性并不是大问题，因为 SEER 具有可解释性，开发人员可以快速检查关注的词元，以验证假阴性。为了理解假阳性的原因，我们从以下几个方面比较了真阴性实例和假阳性实例：

- 测试类型。数据集中的单元测试要么由开发人员编写，要么由 Randoop 自动生成。真阴性实例中的大多数 $t_i$ 都属于 Randoop。然而，对于假阳性，半数 $t_i$ 是开发者编写的测试，另一半是 Randoop 测试。因此，假阳性实例与测试类型之间没有显著相关性。
- # 测试词元 假阳性实例的测试词元平均数量为 74，而真阴性实例为 85，这表明 SEER 在测试较长时表现更好。我们认为，这可能是因为较长测试的表示形式与较短测试不同，使模型更容易为其预测正确的标签。
- # *MUT* 令牌 假阳性实例的 MUT 令牌平均数量为 89，而真负例为 131，这表明当 MUT 的实现包含更多词元和语句时，SEER 的表现更好。与我们关于 # 测试令牌数量一样，较短的 MUT 序列携带的语义信息较少，因此模型更难预测测试结果。
- 缺陷类型。 SEER 能正确预测 95% 的高阶变异体的测试结果。对于 Defects4J 数据集中的真实缺陷，这一比例为 80%。鉴于我们发现假正例和真阴性实例中的缺陷代码行数与结果之间没有显著相关性，我们认为这是因为真实缺陷具有独特性。也就是说，尽管高阶变异会在不同位置注入缺陷，并考虑变异算子的不同组合，但算子的种类有限，因此 SEER 更容易学会涉及这些算子的缺陷。我们认为，这并非 SEER 的局限，而是数据集的问题；通过在训练数据集中纳入更多真实缺陷即可解决这一问题。

## 5.3 研究问题 2：泛化

在前一个研究问题中，我们展示了SEER 能够有效地充当预言机，处理训练数据集中项目中未见过的 $\langle t_i,\ m_i\rangle$配对。在本研究问题中，我们进一步探究 SEER 如何泛化，以预测 $\langle t_i,\ m_i\rangle$ 配对的测试标签，其中 $m_i$所属项目不在训练数据中。为此，我们计算了各项目的贡献度（表1中的"# Tests"列），并将数据集划分为贡献度高于 10%的高贡献项目和低贡献项目。我们使用高贡献数据集中各项目的所有实例进行训练和验证，随后在低贡献数据集中的项目上测试训练好的模型。针对这一研究问题，由于低贡献数据集高度不平衡（通过的 $\langle t_i,\ m_i\rangle$配对数量 3.5× 多于失败的配对），我们仅使用精确率和召回率来评估 SEER的性能。

与 RQ1 中计算的原始精确率和召回率（86% 和 93%）相比，SEER 在未见项目上的性能指标分别为 77%（降低 9% ↓）和 82%（降低 9% ↓）。鉴于未见项目与用于训练的项目具有不同的统计分布，即词元不同，因而词汇表也不同，出现性能下降是意料之中的，这是由分布外（OOD）问题导致的 [64]。我们对误分类实例进行的进一步调查证实，该模型的性能

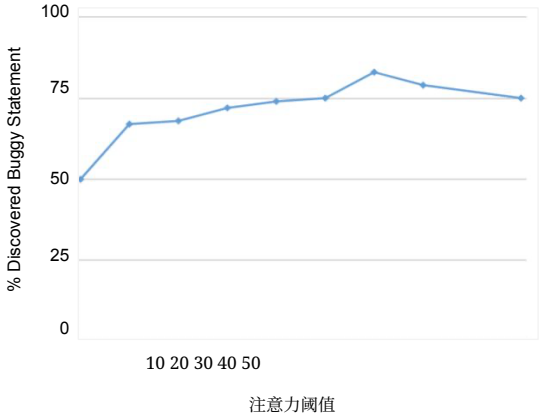
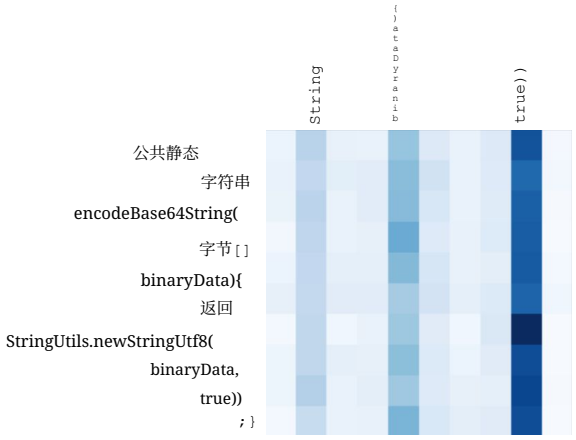



**图 8：受关注的错误语句所占百分比与注意力阈值的关系**




```
public static String encodeBase64String(byte[] binaryData){
    返回 StringUtils.newStringUtf8(
                encodeBase64(二进制数据，真));
}
```

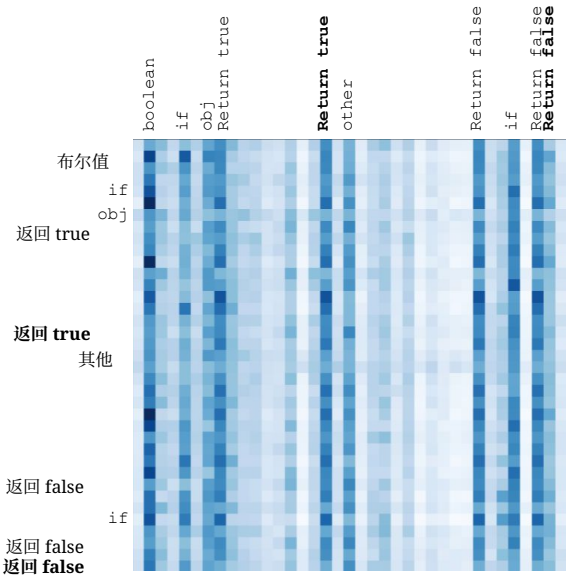
**图9：Codec项目中 encodeBase64String 方法的自注意力可视化，展示了 SEER 对错误标记（即"true"）给予了最高的注意力**

在词汇表与用于训练 SEER 的项目词汇表重合度较高的未见项目上表现更好，而在词汇表重合度较低的项目上则不然。这些结果表明，**SEER 在词汇表与用于训练预言机的项目重合的未见项目上，也能实现相当的性能**。

## 5.4 研究问题3：解释

请回想一下，SEER 的解释有两个目标。首先，我们对嵌入网络进行解释，以验证其是否正确地学会了区分正确的 MUT 和有缺陷的 MUT 的表示。更重要的是，我们对预言机进行解释，以确定在预测标签时，MUT 中的哪些特征（即词元）以及哪些语句最受关注。根据这些信息，我们可以判断受关注的特征是否与决策相关，以及模型的性能是否有效。

5.4.1 注意力分析。给定阈值数量 $k$，SEER 的注意力分析算法（算法2）会生成 MUT 中排名前 $k$% 的受关注词元和受关注语句集合。为确认 SEER 在预测标签时关注了相关词元，我们测量了 MUT 中错误语句属于受关注语句的百分比。具体而言，我们针对真负例测试实例计算了这一指标，即 SEER 正确识别为失败的 $'t_i, m_i($ 对。图 8 展示了错误语句中属于受关注语句的百分比，以及该百分比随阈值变化的情况。结果表明，SEER 确实关注了用于预测测试标签的相关词元；即使阈值仅为 5%，它也能正确识别研究对象 MUT 中 50% 的错误语句。一个有趣的观察是，提高阈值未必能改善缺陷定位。例如，将阈值提高到




```
public 布尔值 equals(对象 对象){
    如果 (此 == 对象)
        返回 true;
    如果 (对象 == 为空 || 获取类() != 对象.获取类())
        返回 true;
    ZIP 归档条目 other = (ZIP 归档条目) obj;
    如果 (名称 == 为空) {
        如果 (other.name != 为空)
            返回 false;
    } 否则如果 (!name.equals(other.name))
        返回 false;
    返回 false;
}
```

**图10：equals方法的自注意力可视化，展示了 SEER 在多个位置关注了有缺陷的词元**

40% 或 50% 会使无缺陷语句也出现在受关注语句中，从而降低受关注语句中错误语句的贡献度。

我们还手动检查了失败的 $\langle t_i, m_i \rangle$配对的自注意力矩阵热力图，以定性地确认 SEER 在预测标签时是否考虑了相关词元。图 9 展示了一个这样的案例，其中有缺陷的 MUT 是 Codec项目中的 `encodeBase64String`。如图 9所示，该缺陷是由于向 `encodeBase64` 方法传入了错误的参数，即"true"而不是"false"。通过查看自注意力矩阵的热力图可视化，我们可以看到，在为这个测试实例预测"失败"标签时，SEER 最关注的是有缺陷的词元（为节省篇幅并便于阅读，我们合并了部分词元，并调整了热力图可视化中的权重）。再来看另一个缺陷更复杂、涉及多个词元或语句的例子，请参见有缺陷的 MUT 及其对应的自注意力热力图 [3] 可视化，如图 10所示。在这个例子中，两个突出显示的 return语句的返回值不正确。查看

[3]这些词元在此热力图中合并显示，且仅标注受关注程度最高的词元。

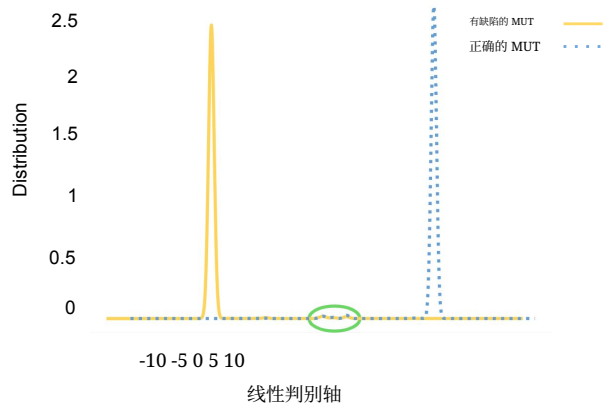



**图 11：线性判别分析（LDA）结果。黄色曲线表示有缺陷的 MUT 的分布，蓝色虚线表示降维后正确 MUT 的分布**

热力图可见，在预测"失败"标签时，这两条语句中的词元都是受关注程度最高的词元。

*5.4.2* 嵌入分析。图11 展示了第4.4.2节所讨论的嵌入分析结果。经过 LDA 降维后，蓝色和黄色分布分别显示了正确 MUT 和有缺陷的 MUT 的嵌入之间的差异。如该图所示，正确 MUT 和有缺陷的 MUT 的分布在低维空间中几乎完全分离，这证实了**SEER能够在嵌入空间中从语义上区分有缺陷和正确 MUT 的表示**。在正确 MUT 和有缺陷的 MUT 分布之间的附近，存在一些重叠的实例 $x = 0$ 。这种重叠表明，在200 维嵌入空间中，少数有缺陷和正确 MUT 的嵌入彼此接近。通过手动检查这些实例，我们发现它们属于$\langle t_i,\ m_i\rangle$即SEER 未能预测出正确标签的成对样本。

## 5.5 研究问题4：性能

为回答这一研究问题，我们评估了阶段1和阶段2训练所需的时间，以及测试预言机所需的时间。实验在配备16GB GDDR6显存的Tesla T4 GPU上运行。批量大小为16对时，阶段1和阶段2训练每个训练轮次的平均耗时分别为447秒和1,745秒。提前终止标准的耐心值设为5个训练轮次，阶段1和阶段2训练分别经过22和30个训练轮次后完成，总训练时间为14小时。鉴于SEER具有泛化能力，模型的一次性训练是合理的。训练完成后，SEER平均仅需6.5毫秒，即可预测给定一对的通过或失败标签。

## 6 相关工作

最先进的测试预言机自动化技术可分为三大类：隐式、指定和派生 预言机。隐式预言机依赖某些隐含知识来判断测试是通过还是失败。这类隐含知识的例子包括缓冲区溢出几乎总会导致错误，过高的 CPU 使用率可能表明服务器发生故障，而不必要的电池用量则是移动应用存在能耗缺陷的证据。隐式预言机虽然相当有效且能够自动运行，但只能检测有限类别的缺陷。

特定预言机确定测试执行的预期输出，并将其与实际输出进行比较，以判断测试是通过还是失败。为了识别预期输出，这类预言机要求被测系统存在形式化规格 [18, 24, 28, 36, 38, 55] 或契约（前置条件和后置条件） [5, 9, 16, 17, 78] 。这类预言机的性能和可用性高度依赖于规格的可用性、完整性和质量。然而，对于许多不断变化的软件系统，规格和契约要么不存在，要么已经过时。即使自动化技术能够生成规格，这些规格通常也相当抽象，因此无法保证从中推断出具体的测试输出，或者推断结果不准确 [6, 27]。最先进的机器学习技术通过预测有意义的规格 [48] 或断言语句 [72, 77, 80]，缓解了这些局限。与SEER相比，这些技术仅在某一时点评估与程序行为相关的一组有限属性，即断言 [18]。

派生预言机 通过区分系统的正确行为与错误行为，而非了解确切输出，来判定测试通过还是失败。正确和错误的行为可以通过以下方式获得：(1) 从 执行日志 [3, 7, 8, 22, 23, 26, 31, 42, 47, 50, 56, 63, 65, 66, 75]等元数据中推断；(2) 作为预期功能的属性提供（蜕变关系） [4, 11, 12, 19–21、43、46、49、57–62, 68, 69, 71, 73, 83, 85, 89, 91, 92]；或 (3) 与软件的其他版本进行比较 [14, 32, 67, 81, 82, 86]。派生预言机具有实用性，但通常并不完备，也就是说，它们只能识别部分输入对应的测试输出。 SEER虽然被视为一种派生预言机，但它通过领域专属嵌入缓解了这一问题，即根据测试结果，在嵌入空间中从语义上区分有缺陷的代码与正确代码。因此，充当预言机的神经模型会利用这些通用 知识来预测测试通过或失败的判定。

## 7 总结

测试预言机自动化一直是软件工程领域最具挑战性的问题之一，但与测试输入生成相比，受到的关注较少。本文提出了SEER，这是一种新颖的深度学习技术，旨在推动自动化测试预言机构建更进一步。SEER 通过从高质量且多样化的数据集中学习输入与输出之间的语义相关性，为给定的 '$t_i$、$m_i$( 预测通过或失败的判定。实验结果表明，所学习的预言机能够准确、高效地预测测试结果，并且能够泛化到训练期间未见过的项目。

目前，我们正在考虑未来工作的几个方向。鉴于我们为代码和测试生成的领域特定表示取得了令人鼓舞的成果，我们将探索这些表示在其他软件分析任务中的应用，例如漏洞检测、缺陷定位和程序修复。此外，我们计划将 SEER 扩展到系统测试。系统测试比单元测试更复杂、规模更大，这可能需要将 SEER 的架构改为图神经网络（GNN），以便更好地捕捉代码语义和表示。

# 参考文献

[1] 2021。Defects4J 数据集。https://github.com/rjust/defects4j。[2] 2022。SEER 工件。https://github.com/Intelligent-CAT-Lab/SEER。[3] Shay Artzi Michael D Ernst Adam 和 Kiezun Carlos Pacheco Jeff H Perkins。2006。大海捞针：为面向对象程序生成合法测试输入。以及面向对象系统（2006），27。[4] John Ahlgren, Maria Eugenia Berezin, Kinga Bojarczuk, Elena Dulskyte, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Erik Meijer 等。2021。利用蜕变测试大规模测试支持 Web 的仿真。载于《2021 IEEE/ACM 第43届国际软件工程会议：软件工程实践》（ICSE-SEIP）。IEEE，140–149。[5] Wladimir Araujo、Lionel C Briand 和 Yvan Labiche。2011。启用 Java 建模语言中并发契约的运行时断言检查。载于《2011年第33届国际软件工程会议》（ICSE）。IEEE，786–795。[6] Earl T Barr、Mark Harman、Phil McMinn、Muzammil Shahbaz 和 Shin Yoo。2014。软件测试中的预言机问题：综述。《IEEE软件工程汇刊》41，5（2014），507–525。[7] Antonia Bertolino、Paola Inverardi、Patrizio Pelliccione 和 Massimo Tivoli。2009。自动合成可组合 Web 服务的行为协议。载于第7届欧洲软件工程会议与 ACM SIGSOFT 软件工程基础研讨会联合会议论文集。141–150。[8] Dirk Beyer、Thomas A Henzinger、Ranjit Jhala 和 Rupak Majumdar。2005。使用 Blast 检查内存安全。载于《软件工程基础方法国际会议》。Springer，2–18。[9] Lionel C Briand、Yvan Labiche 和 Hong Sun。2003。研究分析契约在提高面向对象代码可测试性方面的应用。《软件：实践与经验》33，7（2003），637–672。[10] Richard A Brualdi。1977。组合学入门。印度培生教育。[11] Tsong Yueh Chen、Fei-Ching Kuo、Huai Liu、Pak-Lok Poon、Dave Towey、TH Tse 和 Zhi Quan Zhou。2018。蜕变测试：挑战与机遇综述。《ACM 计算调查》（CSUR）51，1（2018），1–27。[12] Tsong Yueh Chen、Fei-Ching Kuo、Wenjuan Ma、Willy Susilo、Dave Towey、Jeffrey Voas 和 Zhi Quan Zhou。2016。面向网络安全的蜕变测试。《Computer》49，6（2016），48–55。[13] Tsong Yueh Chen、F-C Kuo、TH Tse 和 Zhi Quan Zhou。2003。蜕变测试及其拓展。载于第十一届软件技术与工程实践国际年度研讨会。IEEE，94–100。[14] Tsong Yueh Chen、Pak-Lok Poon 和 Xiaoyuan Xie。2016。METRIC：基于类别选择框架的蜕变关系识别。《系统与软件杂志》116（2016），177–190。[15] Jianpeng Cheng、Li Dong 和 Mirella Lapata。2016。用于机器阅读的长短期记忆网络。arXiv 预印本 arXiv:1601.06733（2016）。[16] Yoonsik Cheon。2007。基于断言的测试预言机中的抽象。载于第七届高质量软件国际会议（QSIC 2007）。IEEE，410–414。[17] Yoonsik Cheon 和 Gary T Leavens。2002。单元测试的简单实用方法：JML 和 JUnit 之道。载于《欧洲面向对象编程会议》。Springer，231–255。[18] David Coppit 和 Jennifer M Haddox-Schatz。2005。关于将基于规范的断言用作测试预言机。载于《第29届 IEEE/NASA 软件工程年度研讨会》。IEEE，305–314。[19] Junhua Ding、Xiaojun Kang 和 Xin-Hua Hu。2017。通过蜕变测试验证深度学习框架。载于《2017 IEEE/ACM 第2届蜕变测试国际研讨会》（MET）。IEEE，28–34。[20] Alastair F Donaldson 和 Andrei Lascu。2016。面向（图形）编译器的蜕变测试。载于第1届蜕变测试国际研讨会论文集。44–47。[21] Anurag Dwarakanath、Manish Ahuja、Samarth Sikand、Raghotham M Rao、RP Ja-gadeesh Chandra Bose、Neville Dubash 和 Sanjay Podder。2018。使用蜕变测试识别基于机器学习的图像分类器中的实现缺陷。载于第27届 ACM SIGSOFT 国际软件测试与分析研讨会论文集。118–128。[22] Michael D Ernst、Jake Cockrell、William G Griswold 和 David Notkin。2001。动态发现可能的程序不变量以支持程序演化。《IEEE软件工程汇刊》27，2（2001），99–123。[23] Michael D Ernst、Jeff H Perkins、Philip J Guo、Stephen McCamant、Carlos Pacheco、Matthew S Tschantz 和 Chen Xiao。2007。Daikon 系统：动态检测可能的不变量。《计算机编程科学》69，1-3（2007），35–45。[24] Xin Feng、David Lorge Parnas、TH Tse 和 Tony O'Callaghan。2011。基于表格表达式的测试策略比较。《IEEE软件工程汇刊》37，5（2011），616–634。[25] Gordon Fraser 和 Andrea Arcuri。2011。EvoSuite：面向对象软件的自动测试套件生成。载于第19届 ACM SIGSOFT 研讨会与第13届欧洲软件工程基础会议论文集。416–419。[26] Kambiz Frounchi、Lionel C Briand、Leo Grady、Yvan Labiche 和 Rajesh Sub-ramanyan。2011。自动化图像分割验证与确认

通过学习测试预言机。《信息与软件技术》53, 12 (2011), 1337–1348。[27] Marie-Claude Gaudel. 2001. 基于形式化规格的测试：一种通用方法。载于《国际可靠软件技术会议》。Springer, 35–48。[28] Gregory Gay、Sanjai Rayadurgam 和 Mats PE Heimdahl. 2014. 通过自动化模型引导提高测试预言判定的准确率。载于第29届ACM/IEEE国际自动化软件工程会议论文集。527–538。[29] Andrej Gisbrecht、Alexander Schulz 和 Barbara Hammer. 2015. 使用核t-SNE进行参数化非线性降维。《神经计算》147 (2015), 71–82。[30] Raul Gomez、Lluis Gomez、Jaume Gibert 和 Dimosthenis Karatzas. 2019. 基于网络数据的多模态检索自监督学习。载于《多模态场景理解》。爱思唯尔, 279–306。[31] Roland Groz、Keqin Li、Alexandre Petrenko 和 Muzammil Shahbaz. 2008. 通过推断、测试和可达性分析进行模块化系统验证。载于《软件与通信系统测试》。Springer, 216–233。[32] Zhongxian Gu、Earl T Barr、David J Hamilton 和 Zhendong Su. 2010. 这个错误真的修复了吗？载于《2010年ACM/IEEE第32届软件工程国际会议》，第1卷。IEEE, 55–64。[33] Shivani Gupta 和 Atul Gupta. 2019. 应对机器学习数据集中的噪声问题：系统综述。《计算机科学Procedia》161 (2019), 466–474。[34] Vincent J Hellendoorn、Charles Sutton、Rishabh Singh、Petros Maniatis 和 David Bieber. 2019. 源代码的全局关系模型。载于《国际学习表征会议》。[35] Reyhaneh Jabbarvand、Forough Mehralian 和 Sam Malek. 2020. 为安卓自动构建能耗测试预言。载于第28届ACM欧洲软件工程会议与软件工程基础研讨会联合会议论文集。927–938。[36] Daniel Jackson. 2012. 《软件抽象：逻辑、语言与分析》。麻省理工学院出版社。[37] Yue Jia 和 Mark Harman. 2008. 使用高阶变异测试构造细微故障。载于《2008年第八届IEEE源代码分析与操作国际工作会议》。IEEE, 249–258。[38] Ying Jin 和 David Lorge Parnas. 2010. 定义表格式数学表达式的含义。《计算机编程科学》75, 11 (2010), 980–1000。[39] Andrej Karpathy 和 Li Fei-Fei. 2015. 用于生成图像描述的深度视觉-语义对齐。载于《IEEE计算机视觉与模式识别会议论文集》。3128–3137。[40] Shachar Kaufman、Saharon Rosset、Claudia Perlich 和 Ori Stitelman. 2012. 数据挖掘中的数据泄漏：表述、检测与规避。《ACM数据知识发现汇刊》6, 4 (2012), 1–21。[41] Diederik P Kingma 和 Jimmy Ba. 2014. Adam：一种随机优化方法。arXiv预印本 arXiv:1412.6980 (2014)。[42] Keqin Li、Roland Groz 和 Muzammil Shahbaz. 2006. 由增量状态机学习引导的组件集成测试。载于《测试：学术与工业会议——实践与研究技术》(TAIC PART'06)。IEEE, 59–70。[43] Xuanyi Lin、Michelle Simon 和 Nan Niu. 2018. 面向科学软件的探索式蜕变测试。《科学与工程计算》22, 2 (2018), 78–87。[44] Ilya Loshchilov 和 Frank Hutter. 2017. 解耦权重衰减正则化。arXiv预印本 arXiv:1711.05101 (2017)。[45] Chunyan Ma、Shaoying Liu、Jinglan Fu 和 Tao Zhang. 2021. 基于反向传播神经网络的程序测试预言生成：利用不同断点处的变量值。《国际软件工程与知识工程杂志》31, 10 (2021), 1469–1494。[46] Pingchuan Ma、Shuai Wang 和 Jin Liu. 2020. 自然语言处理模型中的公平性违规的蜕变测试与认证式缓解。载于国际人工智能联合会议。458–465。[47] Maik Merten、Falk Howar、Bernhard Steffen、Patrizio Pellicione 和 Massimo Tivoli. 2012. 基于接口描述自动推断黑盒系统模型。载于《形式化方法、验证与确认应用国际研讨会》。Springer, 79–96。[48] Facundo Molina、Marcelo d'Amorim 和 Nazareno Aguirre. 2022. 模糊测试类规格说明。arXiv预印本 arXiv:2201.10874 (2022)。[49] Christian Murphy、Gail E Kaiser 和 Lifeng Hu. 2008. 用于蜕变测试的机器学习应用的特性。(2008)。[50] Carlos Pacheco 和 Michael D Ernst. 2005. Eclat：自动生成并分类测试输入。载于《欧洲面向对象编程会议》。Springer, 504–527。[51] Carlos Pacheco 和 Michael D Ernst. 2007. Randoop：针对 Java 的反馈引导随机测试。载于第22届ACM SIGPLAN面向对象编程系统与应用会议伴随会议论文集。815–816。[52] Sinno Jialin Pan 和 Qiang Yang. 2009. 迁移学习综述。《IEEE知识与数据工程汇刊》22, 10 (2009), 1345–1359。[53] Matthew Partridge 和 Rafael A Calvo. 1998. 快速降维与简易主成分分析。《智能数据分析》2, 3 (1998), 203–214。

[54] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gregory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga、Alban Desmaison、Andreas Kopf、Edward Yang、Zachary DeVito、Martin Raison、Alykhan Tejani、Sasank Chilamkurthy、Benoit Steiner、Lu Fang、Junjie Bai 和 Soumith Chintala. 2019. PyTorch：一种命令式风格、高性能深度学习库。载于《神经信息处理系统进展》第32卷，H. Wallach、H. Larochelle、A. Beygelzimer、F. d'Alché-Buc、E. Fox 和 R. Garnett（编）。Curran Associates, Inc., 8024–8035.
h
t
t
p://
p
a
p
e
r
s.
n
e
u
r
i
ps.cc/
paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf [55] Dennis K Peters 和 David Lorge Parnas. 2002. 面向实时系统的基于需求的监视器。《IEEE软件工程汇刊》28, 2 (2002), 146–158。[56] Sam Ratcliff、David R White 和 John A Clark. 2011. 使用遗传编程和变异测试搜索不变量。载于第13届遗传与进化计算年会论文集。1907–1914。[57] Sergio Segura、Amador Durán、Ana B Sánchez、Daniel Le Berre、Emmanuel Lonca 和 Antonio Ruiz-Cortés. 2015. 可变性分析工具的自动化蜕变测试。《软件测试、验证与可靠性》25, 2 (2015), 138–163。[58] Sergio Segura、Gordon Fraser、Ana B Sanchez 和 Antonio Ruiz-Cortés. 2016. 蜕变测试综述。《IEEE软件工程汇刊》42, 9 (2016), 805–824。[59] Sergio Segura、Robert M Hierons、David Benavides 和 Antonio Ruiz-Cortés. 2010. 特征模型分析的自动化测试数据生成：一种蜕变测试方法。载于《2010年第三届软件测试、验证与确认国际会议》。IEEE, 35–44。[60] Sergio Segura、Robert M Hierons、David Benavides 和 Antonio Ruiz-Cortés. 2011. 特征模型分析的自动化蜕变测试。《信息与软件技术》53, 3 (2011), 245–258。[61] Sergio Segura、José A Parejo、Javier Troya 和 Antonio Ruiz-Cortés. 2017. RESTful Web API的蜕变测试。《IEEE软件工程汇刊》44, 11 (2017), 1083–1099。[62] Sergio Segura、Dave Towey、Zhi Quan Zhou 和 Tsong Yueh Chen. 2018. 蜕变测试：测试不可测试之物。《IEEE Software》37, 3 (2018), 46–53。[63] Seyed Reza Shahamiri、Wan MN Wan-Kadir、Suhaimi Ibrahim 和 Siti Zaiton Mohd Hashim. 2012. 人工神经网络作为多网络自动化测试预言机。《自动化软件工程》19, 3 (2012), 303–334。[64] Zheyan Shen、Jiashuo Liu、Yue He、Xingxuan Zhang、Renzhe Xu、Han Yu 和 Peng Cui. 2021. 迈向分布外泛化：一项综述。arXiv预印本 arXiv:2108.13624 (2021)。[65] Rishabh Singh、Dimitra Giannakopoulou 和 Corina Păsăreanu. 2010. 使用可能与必须抽象学习组件接口。载于《计算机辅助验证国际会议》。Springer, 527–542。[66] Matt Staats、Shin Hong、Moonzoo Kim 和 Gregg Rothermel. 2012. 理解用户的理解：确定生成的程序不变量是否正确。载于《2012年国际软件测试与分析研讨会论文集》。188–198。[67] Chang-Ai Sun、An Fu、Pak-Lok Poon、Xiaoyuan Xie、Huai Liu 和 Tsong Yueh Chen. 2019. METRIC⁺+：一种基于输入加输出域的蜕变关系识别技术。《IEEE软件工程汇刊》47, 9 (2019), 1764–1785。[68] Liqun Sun 和 Zhi Quan Zhou. 2018. 机器翻译的蜕变测试：MT4MT。载于《2018年第25届澳大拉西亚软件工程会议》(ASWEC)。IEEE, 96–100。[69] Qiuming Tao、Wei Wu、Chen Zhao 和 Wuwei Shen. 2010. 一种基于蜕变测试技术的编译器自动测试方法。载于《2010年亚太软件工程会议》。IEEE, 270–279。[70] Maurice M Tatsuoka 和 David V Tiedeman. 1954. 第四章：判别分析。《教育研究评论》24, 5 (1954), 402–420。[71] Sandro Tolksdorf、Daniel Lehmann 和 Michael Pradel. 2019. 调试器的交互式蜕变测试。载于第28届ACM SIGSOFT国际软件测试与分析研讨会论文集。273–283。[72] Michele Tufano、Dawn Drain、Alexey Svyatkovskiy 和 Neel Sundaresan. 2020. 使用预训练模型为单元测试用例生成准确的断言语句

Transformer。arXiv预印本 arXiv:2009.05634（2020）。[73] Muhammad Usman、Wenxi Wang 和 Sarfraz Khurshid。2020。Test MC：使用差分测试和蜕变测试测试模型计数器。载于《第35届IEEE/ACM自动化软件工程国际会议论文集》。709–721。[74] Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N. Gomez、Łukasz Kaiser 和 Illia Polosukhin。2017。Attention is All You Need。载于《第31届神经信息处理系统国际会议论文集》（美国加利福尼亚州长滩）（NIPS'17）。Curran Associates Inc.，美国纽约州雷德胡克，6000–6010。[75] Neil Walkinshaw 和 Kirill Bogdanov。2008。推断带有时间约束的有限状态模型。载于《2008年IEEE/ACM第23届自动化软件工程国际会议》。IEEE，248–257。[76] Yu Wang、Ke Wang、Fengjuan Gao 和 Linzhang Wang。2020。利用图区间神经网络学习语义程序嵌入。《ACM编程语言论文集》4，OOPSLA（2020），1–27。[77] Cody Watson、Michele Tufano、Kevin Moran、Gabriele Bavota 和 Denys Poshyvanyk。2020。学习适用于单元测试用例的有意义的断言语句。载于《第42届ACM/IEEE软件工程国际会议论文集》。1398–1409。[78] Yi Wei、Carlo A Furia、Nikolay Kazmin 和 Bertrand Meyer。2011。推断更好的契约。载于《第33届软件工程国际会议论文集》。191–200。[79] Gary M Weiss 和 Haym Hirsh。2000。小析取项的定量研究。AAAI/IAAI 2000，665–670（2000），15。[80] Robert White 和 Jens Krinke。2020。ReAssert：用于断言生成的深度学习。arXiv预印本 arXiv:2011.09784（2020）。[81] Tao Xie。2006。通过回归预言机检查扩充自动生成的单元测试套件。载于《欧洲面向对象编程会议》。Springer，380–403。[82] Tao Xie 和 David Notkin。2005。窥探黑盒内部：通过比较值谱进行回归测试。《IEEE软件工程汇刊》31，10（2005），869–883。[83] Xiaoyuan Xie、Joshua WK Ho、Christian Murphy、Gail Kaiser、Baowen Xu 和 Tsong Yueh Chen。2011。通过蜕变测试测试和验证机器学习分类器。《系统与软件杂志》84，4（2011），544–558。[84] Ran Xu、Caiming Xiong、Wei Chen 和 Jason Corso。2015。联合建模深度视频和组合文本，在统一框架中连接视觉与语言。载于《AAAI人工智能会议论文集》，第29卷。[85] Shin Yoo。2010。随机优化的蜕变测试。载于《2010年第三届软件测试、验证与确认国际会议研讨会》。IEEE，192–201。[86] Andreas Zeller 和 Ralf Hildebrandt。2002。简化并隔离导致失败的输入。《IEEE软件工程汇刊》28，2（2002），183–200。[87] Jie Zhang、Junjie Chen、Dan Hao、Yingfei Xiong、Bing Xie、Lu Zhang 和 Hong Mei。2014。基于搜索的多项式蜕变关系推断。载于《第29届ACM/IEEE自动化软件工程国际会议论文集》。701–712。[88] Lingming Zhang、Guowei Yang、Neha Rungta、Suzette Person 和 Sarfraz Khurshid。2014。反馈驱动的动态不变量发现。载于《2014年国际软件测试与分析研讨会论文集》。362–372。[89] Mengshi Zhang、Yuqun Zhang、Lingming Zhang、Cong Liu 和 Sarfraz Khurshid。2018。DeepRoad：面向自动驾驶系统的基于生成对抗网络的蜕变测试与输入验证框架。载于《2018年第33届IEEE/ACM自动化软件工程国际会议》（ASE）。IEEE，132–142。[90] Pengcheng Zhang、Xuewu Zhou、Patrizio Pelliccione 和 Hareton Leung。2017。RBF-MLMR：一种使用径向基函数神经网络的多标签蜕变关系预测方法。《IEEE Access》5（2017），21791–21805。[91] Zhi Quan Zhou 和 Liqun Sun。2019。无人驾驶汽车的蜕变测试。《ACM通讯》62，3（2019），61–67。[92] Zhi Quan Zhou、Shaowen Xiang 和 Tsong Yueh Chen。2015。面向软件质量评估的蜕变测试：搜索引擎研究。《IEEE软件工程汇刊》42，3（2015），264–284。