#### 变异测试应用研究综述

**摘要：**

**关键词：**

**1.引言**

（变异测试简介）

（内容简介）

**2.文献检索**

（检索方法）

（文献筛选）

（文献分类）

**3.变异测试应用**

3.1优化回归测试：

回归测试是软件开发和维护过程中保障软件质量的一种重要手段，用于保障代码修改的正确性，并避免代码修改对被测程序的其他模块造成副作用。目前常见的回归测试技术包括：失效测试用例的识别和修复技术、测试用例选择技术、测试用例优先级排序技术、测试用例集约简技术和测试用例集扩充技术等。

变异测试的主要作用在于评估已有的测试用例在被测程序上的充分性。一个变异体(mutant)就是在原被测程序中人为地用变异算子（mutation operator）引入差错，以生成相对于原程序的变异。如果一个测试用例在测试原被测程序和测试变异程序时结果不一致，那么我们说这个测试用例“杀死”了这个变异体；如果一个测试用例未能“杀死”变异体，可能就要考虑这个测试用例对于被测程序的充分性来说是不足的。一个测试用例的充分性得分就是它能够杀死的变异体的数量。

变异测试可以用于优化回归测试。

Goran等人1考察了一个使用突变测试的工业应用程序，涉及3万多名开发人员和190万更改集，用4种编程语言编写。他们指出，使用生产突变的突变测试不会给软件开发过程增加显著的开销，并且报告了开发人员所感知到的突变测试的好处。他们还量化了非有效的突变的成本，结果表明，实现突变充分性既不实际也不理想。Rudolf等人2调查了突变测试在提高安全关键软件系统中单元测试质量方面的适用性和实用性。在改进测试时，在代码中发现了两个新的错误。针对突变体的测试执行需要超过4000小时的计算时间，总的时间大约是半年。他们所研究系统的一个重要特性是它在单元测试中100%的MC/DC覆盖率，这是安全标准所推荐的。这一覆盖水平满足认证要求。他们调查了突变测试在此背景下的适用性和有用性，以评估现有测试的故障检测有效性，并识别测试中的缺陷。他们还调查并量化了在研究的真实世界软件系统中应用突变测试所涉及的工作和成本。他们的结果表明，突变分析对于改进安全关键软件的验证是潜在的有用的。突变测试能够评估已经达到100%覆盖率的测试套件的质量。此外，突变测试提供了测试用例中很难发现的缺陷的提示。这些反馈可以直接用于修改和增强测试。

突变测试依赖于确定性测试行为(对每个突变取模)来确定突变是否被杀死。August Shi等人3发现，即使测试始终通过，它们仍然可以在代码覆盖级别上表现出细粒度的不确定性，导致测试不确定地执行突变的代码。这些因素可以解释目前现有的突变测试工具无法检测到的总体突变评分中(平均)4个百分点的变化。他们提出了通过改进的覆盖收集来检测和管理突变测试中的这种flakiness的技术，以确保从测试到它们所覆盖的突变的更可靠的映射，并在突变没有覆盖它们时重新运行测试，直到获得可靠的结果。他们的实验结果表明，这种技术将未知突变体的比例降低了79.4%。

测试用例优先级(TCP)是回归测试的一个重要组成部分，允许更早地检测到错误或帮助减少测试时间和成本。虽然以往研究中存在几种TCP方法，但越来越多的研究对它们进行了针对合成软件缺陷(称为突变体)的评估。因此，目前还不清楚在多大程度上，突变体上的TCP性能可以代表在实际故障上实现的性能。为了回答这个问题，Qi等人4 进行了一次实证研究，比较TCP技术应用于现实世界和突变故障的性能。他们研究的背景包括八个经过充分研究的TCP方法，35k+突变错误，以及在defect ts4j数据集中来自五个Java系统的357个真实的错误。他们的研究结果表明，所研究的TCP技术在突变体上的相对性能可能与在真实故障上的性能没有很强的相关性，这取决于主题程序的属性。这表明，在某些情况下，在一组突变体上表现最好的技术在实际应用到实际故障时可能并不是最好的技术。他们还说明了这些相关性对于由不同算子产生的突变体是不同的，这取决于所选择的算子是否反映了主题程序的典型错误。这突出了为特定程序域开发突变运算符的重要性，尤其是对TCP而言。

针对测试用例优先级，Donghwan等人5  提出了一种新的测试用例优先化技术，它结合了基于变异和基于多样性的方法。他们的基于多样性感知的突变技术依赖于突变区分的概念，该概念旨在将一个突变的行为与另一个突变区别开来，而不是与原始程序区别开来。他们以352个实际故障和553477个开发者编写的测试案例，实证研究了基于突变的优先排序技术（即，使用传统的突变杀灭和拟定的突变区分）的相对成本和有效性。实证评估考虑了各种环境下的传统和多样性感知突变标准：单目标贪婪、混合和多目标优化。 结果表明，基于突变的优先级排序至少在88.9%和77.8%的故障上比随机优先级排序和基于覆盖的优先级排序更有效或同样有效。在贪婪、混合和多目标优化策略中，使用仅杀死和多样性意识突变充分性标准，没有一个更好的测试用例优先化技术。有趣的是，虽然仅杀死突变和多样性意识突变充分性标准之间没有优势，但它们的联合使用提高了优先级排序的有效性。对于多目标优化问题，Pareto前沿排序的有效性与排序目标之间没有稳定的相关性。多目标技术的优先级执行时间大约需要37分钟，而贪婪和混合技术需要不到8秒。他们的研究结果有几点启示。例如，尽早识别和杀死突变体比尽早覆盖语句更有效。

将变异用于优化回归测试的案例已经屡见不鲜。比如Coles等人6 开发了Java使用的变异测试工具PIT。PIT十分迅速，因为它直接操纵字节码，并且优化了变异体的执行。它也十分的鲁棒并且在生产工具中得到了很好的继承，它可以在命令行界面、ANT或者MAVEN这样的包管理器中被调用。Just等人7开发了Java的变异测试工具Major。Major提供了集成于OpenJDK编译器中的变异器，并且为JUnit提供了变异测试的分析工具。Major支持常见的变异算子，如二元运算符置换、单元运算符置换（取反等）、常量置换（数值型和字符串型）、分支条件操纵、删除语句（比如删除一行函数调用）。

基于JavaScript的变异测试工具相对较少，对于NodeJS后端运行环境的变异测试工具更少了。但是随着NPM的使用者越来越多、其上的模块也越来越多，一个模块上的bug可能会导致巨大的影响。Rodríguez-Baquero等人8 开发了面向NodeJS的变异测试工具，并将其用到了NPM最热门的且自带测试用例的前20个模块中的12个。

也有研究指出了当前常见的变异测试工具的不足，并提出了改进。Kintis等人9对Java的三种变异测试框架进行了比较。也测量了生成的等价变异体的百分比。得出的结论是Major，MuJava 和 PIT 均不能包含彼此，即每个工具都还有改进的余地，主要是通过扩展它们所支持的变异种子。测试结果表明，这三种工具并不像预期地那样有效。这也揭示了关于变异测试的理论研究日渐成熟，但是常见的变异测试工具并不能满足研究范式的事实。Laurent等人10指出，大部分的变异测试工具仅使用了少量的且严格的变异算子。由于变异测试的可靠性很大程度上依赖于变异算子的选择，因此这样的现象是令人担忧的。他们使用了扩展的变异算子然并在PIT上进行了实现。在真实的项目中，PIT原有的变异算子产生的变异体更容易被杀死，它们平均比扩展的变异算子生成的评分低了35%到75%。

3.2引导测试生成：

在软件测试中，测试用例可以定义为一组输入，并开发执行先决条件来验证特定需求是否得到满足，能够满足测试目标的测试用例集称为有效测试套件。手动生成测试用例是一项复杂而又昂贵的任务，自动测试用例生成被认为是手动测试用例生成的一种替代方法，因为它可以减少过程中涉及的成本和时间。

如何生成高效的变异体和测试用例是变异测试研究中的一大方向。变异测试存在一大难点，即难以甄别出与被测程序等价的变异体。等价的变异体是指那些在语法上与被测程序不同，但是在语义上与被测程序相同的变异体。这些变异体不能被测试用例杀死（因为测试用例在等价变异体和被测程序上的行为是一致的，因此无法杀死这些变异体），但是还是要占据相当多的时间和成本去执行这些等价的变异体。Mirshokraie等人11指出，在过往的研究中发现，变异体生成阶段生成的变异体中，生成的10%~40%的变异体均与原程序等价。识别出等价变异体是一项繁杂耗时的人工工作，并且过往的研究集中在如何在变异体生成阶段之后发现等价变异体。Mirshokraie等人提出了一套通用的先验性规则，用于指导变异体生成器生成更少的等价变异体。简单来说，就是结合静态和动态程序分析技术，找出那些更易出错的代码和对程序行为有着显著影响的代码，并优先在这些代码上生成变异体。

变异测试生成的另一大难点是生成的变异体数量庞杂，测试成本高昂。Zhang等人12使用对变异体设置优先级和精简用到的变异体的方法，以快速地找出被杀死的和未被杀死的变异体，从而降低了所需生成的变异体的数量。

在过去，大多数关于变异测试的研究工作都集中在寻找非科学软件中的缺陷。所提出的方法旨在借助生物启发算法来改进过程，以识别科学代码中的变异体。科学软件依赖于输出的准确性而不是正确性。因此，一般的变异测试不足以发现错误，因为科学软件即使对小的舍入误差也很敏感。Nishtha等人13提出一种新的算法。他们使用的基于搜索的技术是一种改进的Crow Search Algorithm（CSA）用于生成测试数据。 在CSA中，搜索模式取决于称为飞行长度的独立参数。这可能导致搜索解决方案被困在本地区域内的问题。因此，提出的改进的ICSA方法为每个测试用例生成柯西随机数，以防止搜索陷入局部搜索，因此解决了CSA中的一些限制。

3.3缺陷定位：

在整个调试活动中，缺陷定位一直是一个代价昂贵的阶段。传统的人工定位需要测试者努力理解被测程序的复杂内部逻辑，以及测试通过和失败的运行之间的差异，这无疑是一个耗时耗力的工程，特别是对经验不足的开发者而言。

由此，自动缺陷定位技术得到了广泛的研究，目前最流行的方法是基于频谱的缺陷定位（SBFL）。该方法使用程序谱（即测试套件执行的信息概要），根据预测的包含缺陷的风险对程序语句进行排序。然后，开发人员将按照给定排名中语句的顺序检查被测程序，希望在排名顶部附近会遇到错误语句。但是这样的方法消耗较多程序运行时间成本和资源成本，且测试用例的数量和质量对缺陷定位性能影响较大。此外，此方法因为使用的模型（排名的线性检验）不具备足够的现实性和较低的准确性，也一直受到批评。

因此，有很多人开始考虑其他方法的缺陷定位，基于变异测试思想的缺陷定位正是目前许多人的研究方向。

Mike Papadakis等人14考虑了一种基于变异分析的缺陷定位方法，通过人为地向代码中注入缺陷来探索程序的行为，并获得一些可用于自动故障定位的信息，该方法与现有的基于频谱的缺陷定位方法进行比较，结果显示明显优于后者。又由于应用变异分析需要消耗大量的计算资源，他们将选择性变异技术应用到缺陷定位问题，并确定了故障定位背景下具有代表性的变异算子集，来减少变异体数量。最终实验的结果表明，该方法能够将涉及的突变体数量减少80%，而不会损失故障定位精度。

Mike Papadakis等人15在另一些研究中提出了一种基于变异分析的故障定位方法——Metallaxis，其创新之处在于它使用了变异体，并将它们与错误的程序位置联系起来。因此，大部分因测试失败而死亡的变异体提供了一个关于故障位置的良好指示。使用Metallaxis的实验表明，它比基于语句的方法更有效。即使在有助于降低变异成本的技术（如变异采样）的情况下，这也是正确的。此外，受控实验的结果表明，使用变异作为测试技术有利于缺陷定位过程。因此，通过使用基于变异的测试套件而不是基于块或基于分支的测试套件，可以显著改进缺陷定位。他们还证明了Metallaxis具有良好的可扩展性。

Seokhyeon Moon等人16提出了一种新的基于变异测试的故障定位技术——MUSE，MUSE使用变异分析来唯一地捕捉单个程序语句和观察到的故障之间的关系，而不受来自块结构的共享排名的强制影响。其基本思想是，由于错误的程序可以通过修改错误的语句来修复，因此修改错误的语句将使更多失败的测试用例通过。相比之下，由于改变正确的语句会引入新的错误语句，因此改变正确的语句将使更多通过的测试用例失败。

Seokhyeon Moon等人还为缺陷定位技术提出了一种新的评估指标，LIL（位置信息损失），其利用信息理论，通过定位技术测量断层的真实位置和预测位置之间的信息损失。LIL可以应用于任何故障定位技术，并描述任意数量缺陷的定位。最终，使用传统的消耗指标和提出的LIL指标，针对五个真实世界程序的14个错误版本评估MUSE。结果表明，平均而言，MUSE比目前最先进的SBFL技术Op2准确约25倍。

通过对这些文献的深入分析，可以发现将变异分析引入缺陷定位问题，在准确性上都明显优于目前主流的基于频谱的缺陷定位技术（SBFL）。然而两种方法都会消耗大量的计算资源，对于基于变异分析的缺陷定位，这是由于变异测试本身要求生成大量变异体（其中大部分都为无效的等价变异体）所造成的，因此研究人员也在积极探究使用各种技术，如使用选择性变异技术，来降低变异分析的资源消耗。

3.4 AI测试：

AI技术在近十年来得到了迅速的发展，并且在许多领域都得到了应用，然而对于AI的测试与经典的软件测试差别较大。经典的程序设计中，输入的是规则（程序）和需要这些规则处理的数据，系统输出的是答案。因此，经典的软件测试，我们可以通过对比答案来验证和确认程序的正确性。然而，数据驱动的AI系统，不是通过编写明确的逻辑，而是通过数据来训练程序，输入是数据和从这些数据中预期得到的答案，AI系统输出的是规则，这些规则可应用于新的数据，自动计算出答案。可见，数据驱动的AI系统是一种新的编程范式，这种编程范式给AI测试带来了极大的挑战。

具体而言，AI测试面临的挑战有（1）AI系统输出结果很难预测；（2）AI系统测试通过的准则很难确定；（3）AI 系统的输出结果随时间变化；（4）AI系统需要更高效的持续测试方法；（5）AI系统的性能依赖于数据。

AI系统对数据的依赖性极高，而变异测试正是通过变异算子生成大量的变异体（变异体为程序或测试数据），因此许多研究人员想到将变异测试的思想应用到AI测试之中，并提出了很多新的方法。

Lorenz Klampf等人17研究了变异测试在神经网络背景下的适用性。他们考虑了不同的神经网络实现，并利用可用的变异测试工具进行了实验评估，以检查是否可以使用普通的变异测试（即未用于神经网络的变异测试）来获得关于测试这些神经网络实现的程度的有价值信息。特别是，利用了神经网络的评估方法，其中利用可用的测试数据来计算变异分数，即检测到的变异和神经网络库实现中引入的所有变异的分数。此外，还研究了测试数据的大小对变异得分的影响。Lorenz Klampf等人进行了考虑MLP和卷积神经网络的实验，表明变异分数不足以找到足够数量的变异体。此外，还扩展了实验，涵盖了不同大小的测试数据集的情况，也表明测试数据不足以测试神经网络。最终，得出结果为有必要为神经网络库设计专门的测试套件，以大幅提高突变分数。

Lei Ma等人18提出了一个专门用于深度学习（DL）系统的突变测试框架，用以测量测试数据的质量。为此，Lei Ma等人首先定义了一组源级变异算子，以向DL源（即训练数据和训练程序）注入故障。然后，又设计了一组模型级变异算子，该算子在没有训练过程的情况下直接将故障注入DL模型。测试数据的质量可以通过对注入故障的检测程度的分析来评估（为此还提出了两种DL特异性突变测试指标）。最终，在两个公共数据集（MNIST和CIFAR-10）和三个DL模型上证明了所提出的变异测试技术的有用性。

Qiang Hu等人19提出了个用于前馈神经网络（FNN）和递归神经网络（RNN）的变异测试框架——DeepMutation++。DeepMutation++为DeepMulation中引入的FNN模型引入了八个模型级算子，并进一步提出了九个专门用于RNN模型的新算子。特别是，为了迎合RNN的特点，DeepMutation++支持静态变异生成以整体分析测试数据，以及动态变异生成以在运行时检测测试输入的脆弱片段。与深度神经网络（DNN）的结构覆盖标准不同，DeepMutation++能够直接提供关于DNN对输入的鲁棒性的反馈。 最终，Qiang Hu等人证明了DeepMutation++在DNN稳健性分析和测试数据脆弱片段检测的两个典型场景中的有用性：1）基于MNIST数据集的FNN（即LetNet-5）图像处理，2）基于IMDB数据集的RNN（即LSTM和GRU）文本情感分析。还发现，基于变异测试定义的度量可以是DNN鲁棒性的一个重要指标，与DNN抵抗对抗性攻击的鲁棒性有很强的相关性。

在主动自动机学习中存在着两种查询，成员查询和等价查询,其中成员查询对于学习黑盒系统很容易实现。等价查询则更难实现，一些研究提出了一致性测试算法，但这些算法存在两个问题，一是要求要求黑盒系统的状态数有一个固定的上限（通常是未知的），二是构建的测试套件的大小在这个范围内是指数级的。因此实现等价预言可以被视为自动机学习的真正瓶颈。Bernhard K. Aichernig等人20提出了一种基于变异测试的等价查询的实现，更具体地说是基于模型的变异测试，其基于新的目标不是试图证明等价性，而是快速找到反例的思想，尽量减少执行等价查询的测试次数。实验中将随机测试与变异分析相结合，以实现测试的高变异性，并适当解决覆盖率问题。为了证明其有效性，本实验将该方法应用于学习实验，并将其性能与成熟的测试技术——部分W方法进行比较。该评估表明，该方法显著降低了学习成本。在多次实验中，将成本降低了至少一个数量级。

变异测试思想在AI领域的另一个应用是用来检测DNN中的对抗样本攻击。研究表明，即使是训练有素的DNN也容易受到对抗样本的攻击，尤其是当DNN应用于分类任务时。对抗样本通常是通过对抗性扰动来制作的（即对原始样本进行细微修改，使DNN模型对样本进行错误标记），而这样的制作很容易完成。

现有的防御策略主要集中于增加攻击者制作对抗样本难度和基于训练辅助模型检测对抗样本两个思路进行，然而这些防御策略依赖于可用的对抗样本，因此通常仅限于防御特定攻击。

Jingyi Wang等人21发现在DNN上施加随机变异，对抗样本比正常样本敏感得多。于是他们提出了一个灵敏度度量，用实证表明正常样本和对抗样本具有可区分的灵敏度。然后，整合了统计假设检验和模型变异检验，通过测量其灵敏度来检查输入样本在运行时是否可能是正常或敌对的。他们在MNIST和CIFAR10数据集上评估了该方法。结果表明，该方法可以有效、准确地检测由最先进的攻击方法生成的对抗样本。

Jingyi Wang等人22在另一篇研究中提出了nMutant，一种在运行时有效检测对抗样本的方法。通过对MNIST和CIFAR10数据集的实证研究，表明该方法对许多现有的攻击方法（例如FGSM、C&W、JSMA和Blockbox）是有效的。并且该方法工作时不需要任何底层DNN系统的知识，因此可以潜在地应用于广泛的系统。与现有的防御策略相比，它具有合理的可扩展性和可靠性。

3.5其他应用：

变异测试或变异测试思想在其他领域也有着广泛应用，包括对一些较为冷门的或未来的语言的程序测试。

Arzu Behiye Tarımcı等人23针对 PL/SQL (Procedural Language/Structured Query Language)，这一业界采用的但很少被研究关注的动态编程语言，设计了一种变异测试工具，muPLSQL，来帮助变异的生成和测试执行的自动化。具体的，他们提出了44个适用于PL/SQL的变异运算符，其中17个用于PL，21个用于SQL，6个专门用于PL/SQL。同时还将此工具设计为可扩展的，用于合并新的变异运算符。为了评估该工具和变异测试的适用性和有用性，本实验还进行了一个工业案例研究，使用muPLSQL对业务支持软件系统进行变异测试和分析。研究中使用了系统的19个对象，共包含了8206行PL/SQL代码。最终muPLSQL总共产生了5939个变异体，存活变异体数量为680，PL/SQL特异性变异算子产生了320个变异体，其中46个在测试执行中存活。对活变种的手动检查发现，有112个缺失的测试场景和数据验证应纳入现有测试套件。此外，在检查过程中发现了8个源代码错误。

Lorena Gutiérrez-Madroñal等人24将进化变异测试（EMT）应用于Esper EPL编程语言。测试这种编程语言的重要性是不可低估的，它非常适合事件驱动的物联网系统。考虑到许多物联网系统的生存或死亡取决于其对事件的快速反应，测试系统能否根据需求触发预期响应是至关重要的。为了将EMT与Esper EPL结合使用，使用GAmera工具分析了该技术的行为。GAmera是一个包含应用EMT的遗传算法的工具。由于遗传算法的实现，该算法可以在不同的编程语言中使用。开发了一个桥接器来连接涉及的系统:MuEPL和GAmera。根据这两种系统的特点实现了桥。这意味着由于GAmera的适应性，它可以与任何编程语言一起使用。

Pablo G´omez-Abajo等人25提出了一个名为WodelTest的框架，来减少创建变异测试工具的工作量。为了达到这个目的，它遵循了一种模型驱动的方法，在高级描述中合成变异测试工具。该描述使用领域特定语言Wodel来定义和执行模型突变。Wodel是独立于语言的，因为它允许为元模型定义的任何语言创建突变操作符。WodelTest通过生成一个变异测试环境，将被测程序解析为一个模型，应用突变操作符，并根据生成的突变对测试套件进行评估，提供了丰富的变异测试度量集合。测试报告了基于为Java和Atlas转换语言创建变异测试工具的方法评估。

Daniel Fortunato等人26则开始研究变异测试在量子计算领域可能的应用。尽管在经典计算领域，测试已经得到了广泛的研究，并提出了许多方法和工具，但量子程序（QPs）的此类方法仍处于起步阶段，大多数更容易在反直觉的量子编程中犯错误。其中一个问题就是，QPs必然是概率性的，不可能在不中断执行或不影响其结果的情况下进行检查。因此，在量子计算领域，确保QPs的正确实现更加困难。

通过重点研究最流行的用于量子计算的开源全栈库，IBM的量子信息软件包（Qiskit），Daniel Fortunato等人利用语法等效门的概念，提出了五个为QPs定制的新颖的变异算子；以及一个新的基于Python的可以自动执行QPs的变异测试的工具集，名为QMutPy，并且完成了其在24个真实QPs上的有效性和效率的实证评估；同时还详细讨论了如何扩展QPs测试套件以杀死更多的变异体，从而检测更多的bug。实验结果表明，QMutPy可以生成揭示错误的量子变异体，并暴露在实验中使用了真实QPs的测试套件中出现的一些问题。

Jose Miguel Rojas等人27则为变异测试进行了游戏化，开发了一个在线游戏——CODE DEFENDERS，旨在让学习和应用变异测试变得有趣。在游戏中，玩家扮演攻击者的角色，目的是创建最微妙的非等价突变体；或者是防御者的角色，目的是创建强大的测试来杀死这些突变体。这种方法的好处是多方面的:游戏可以发挥教育作用，让学习者以一种有趣的方式参与突变测试活动。有经验的玩家将创造出强大的测试套件，能够检测出其他玩家能够想到的最微妙的漏洞。

变异测试应用的一个具有挑战性的软件领域是移动应用。虽然移动设备和随之而来的应用程序已成为现代计算的支柱，但其开发中使用的框架和模式使测试和验证变得尤为困难。作为帮助衡量和确保移动测试实践有效性的一步，Kevin Moran等人28引入了MDroid+，一个用于Android应用程序变异测试的自动化框架。MDroid+包括来自十种经验衍生的Android故障类型的38个突变算子，自动化检测潜在突变位置和生成突变的过程，并通过可扩展的体系结构促进新操作符的添加和现有操作符的维护。对MDroid+与其他流行的Java语言突变测试工具进行了评估，结果显示，MDroid+生成的不可编译和微不足道的突变更少。

**4.总结：**

（总结前文）

（现有研究的不足）

（对未来研究的看法）

**参考文献**

[1] G. Petrovic, M. Ivankovic, B. Kurtz, P. Ammann and R. Just, "An Industrial Application of Mutation Testing: Lessons, Challenges, and Research Directions," 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2018, pp. 47-53, doi: 10.1109/ICSTW.2018.00027.

[2] Rudolf Ramler, Thomas Wetzlmaier, and Claus Klammer. 2017. An empirical study on the application of mutation testing for a safety-critical industrial software system. In Proceedings of the Symposium on Applied Computing (SAC '17). Association for Computing Machinery, New York, NY, USA, 1401–1408.

[3] Shi A, Bell J, Marinov D. Mitigating the effects of flaky tests on mutation testing[C]//Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2019: 112-122.

[4] Luo Q, Moran K, Poshyvanyk D, et al. Assessing test case prioritization on real faults and mutants[C]//2018 IEEE international conference on software maintenance and evolution (ICSME). IEEE, 2018: 240-251.

[5] Shin D, Yoo S, Papadakis M, et al. Empirical evaluation of mutation‐based test case prioritization techniques[J]. Software Testing, Verification and Reliability, 2019, 29(1-2): e1695.

[6] Coles, Henry, et al. "Pit: a practical mutation testing tool for java." Proceedings of the 25th international symposium on software testing and analysis. 2016.

[7] Just, René. "The Major mutation framework: Efficient and scalable mutation analysis for Java." Proceedings of the 2014 international symposium on software testing and analysis. 2014.

[8] Rodríguez-Baquero, Diego, and Mario Linares-Vásquez. "Mutode: generic javascript and node. js mutation testing tool." Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2018.

[9] Kintis, Marinos, et al. "Analysing and comparing the effectiveness of mutation testing tools: A manual study." 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2016.

[10] Laurent, Thomas, et al. "Assessing and improving the mutation testing practice of pit." 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). IEEE, 2017.

[11] Mirshokraie, Shabnam, Ali Mesbah, and Karthik Pattabiraman. "Efficient JavaScript mutation testing." 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation. IEEE, 2013.

[12] Zhang, Lingming, Darko Marinov, and Sarfraz Khurshid. "Faster mutation testing inspired by test prioritization and reduction." Proceedings of the 2013 International Symposium on Software Testing and Analysis. 2013.

[13] Jatana, Nishtha; Suri, Bharti (2020). An Improved Crow Search Algorithm for Test Data Generation Using Search-Based Mutation Testing. Neural Processing Letters, (), –. doi:10.1007/s11063-020-10288-7

[14] Papadakis M, Le Traon Y. Effective fault localization via mutation analysis: A selective mutation approach[C]//Proceedings of the 29th annual ACM symposium on applied computing. 2014: 1293-1300.

[15] Papadakis M, Le Traon Y. Metallaxis‐FL: mutation‐based fault localization[J]. Software Testing, Verification and Reliability, 2015, 25(5-7): 605-628.

[16] Moon S, Kim Y, Kim M, et al. Ask the mutants: Mutating faulty programs for fault localization[C]//2014 IEEE Seventh International Conference on Software Testing, Verification and Validation. IEEE, 2014: 153-162.

[17] L. Klampfl, N. Chetouane and F. Wotawa, "Mutation Testing for Artificial Neural Networks: An Empirical Evaluation," 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), 2020, pp. 356-365, doi: 10.1109/QRS51102.2020.00054.

[18] Ma L, Zhang F, Sun J, et al. Deepmutation: Mutation testing of deep learning systems[C]//2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2018: 100-111.

[19] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu and J. Zhao, "DeepMutation++: A Mutation Testing Framework for Deep Learning Systems," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1158-1161, doi: 10.1109/ASE.2019.00126. [20] Aichernig B K, Tappler M. Efficient active automata learning via mutation testing[J]. Journal of Automated Reasoning, 2019, 63(4): 1103-1134.

[21] Wang J, Dong G, Sun J, et al. Adversarial sample detection for deep neural network through model mutation testing[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 1245-1256.

[22] Wang J, Sun J, Zhang P, et al. Detecting adversarial samples for deep neural networks through mutation testing[J]. arXiv preprint arXiv:1805.05010, 2018.

[23] Tarımcı A B, Sözer H. Mutation testing of PL/SQL programs[J]. Journal of Systems and Software, 2022, 192: 111399.

[24] Gutiérrez‐Madroñal L, García‐Domínguez A, Medina‐Bulo I. Evolutionary mutation testing for IoT with recorded and generated events[J]. Software: Practice and Experience, 2019, 49(4): 640-672.

[25] Gómez-Abajo P, Guerra E, Lara J, et al. Wodel-Test: a model-based framework for language-independent mutation testing[J]. Software and Systems Modeling, 2021, 20(3): 767-793.

[26] Fortunato D, Campos J, Abreu R. Mutation Testing of Quantum Programs: A Case Study With Qiskit[J]. IEEE Transactions on Quantum Engineering, 2022, 3: 1-17.

[27] Rojas J M, Fraser G. Code defenders: a mutation testing game[C]//2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2016: 162-167.

[28] Moran K, Tufano M, Bernal-Cárdenas C, et al. Mdroid+: A mutation testing framework for android[C]//2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). IEEE, 2018: 33-36.