基于语言、领域应用的变异测试应用文献综述

211870018 陈宇航 211870018 刘子恒 211870018 侍宇 211870018 肖安喆

摘 要 本文对变异测试的研究现状和潜在发展方向进行了综述，通过综述协议筛选并分类了 30 篇符合条件的文献。在变异测试的技术细节基础上，提出了四个具体的研究问题，探讨了变异测试在不同领域、不同语言的应用现状。在各研究问题中我们发现总结了亮点。在应用领域方面，我们发现变异测试在 AI 领域的应用呈上升趋势，有较大研究空间。在语言应用方面，我们发现变异算子的生成过程有优化空间和较多的创新应用。文章还特别关注了变异测试在 AI 领域的应用。最后，我们对已有研究中存在的问题进行了总结，主要集中在变异体生成和质量、大规模软件系统的可扩展性、整合到软件开发流程的挑战等方面。通过对这些问题的分析，为未来变异测试应用领域的研究方向提供了启示。希望本文能够促进变异测试在不同领域的更广泛、更有效的应用。

关键词 变异测试、应用、文献综述、软件开发、人工智能

Literature Review on Language and Domain Application of Mutation Testing

CHEN Yu-Hang LIU Zi-Heng Shi-Yu XIAO An-Zhe

Abstract

This paper provides a comprehensive review of the current state and potential directions of research in mutation testing. Thirty relevant articles were systematically selected and categorized through a review protocol. Building upon the technical details of mutation testing, four specific research questions were formulated, investigating the application status of mutation testing in different domains and languages. Highlights were summarized for each research question. In the realm of applications, a rising trend of mutation testing in the field of AI was observed, indicating significant research opportunities. Concerning language applications, optimization possibilities and innovative uses were identified in the generation process of mutation operators. The paper pays particular attention to the application of mutation testing in the AI domain. Finally, an overview of existing challenges in current research was presented, focusing on issues such as mutation generation and quality, scalability in large software systems, and challenges in integration into the software development process. Through the analysis of these challenges, insights were provided for future research directions in the application of mutation testing. It is hoped that this paper will promote a wider and more effective application of mutation testing in diverse domains.

Keywords Mutation Testing; Application; Literature Review; Software Development; Artificial Intelligence

1 引言

为了深入了解变异测试的应用研究现状，我们开展了本次综述工作。首先我们制定了一项详细的综述协议，筛选后得到 30 篇符合协议的文献，并进行了分类。在文章的第二部分，我们从变异测试的技术细节出发，提出四个具体的研究问题（RQ）。在文章第五部分，系统性地探讨变异测试的应用现状、在人工智能与传统领域的差异，以及在应对应用测试过程中面临的不足和挑战等方面的问题。最终，我们通过对综述工作的全面总结，反思了自身工作的不足之处，并探讨了未来可能的研究方向。

2 变异测试

2.1 基础概念

变异测试的原理主要基于对源代码进行有意的、微小的改动，这些改动被称为“变异”(Mutations)，其核心目标是评估测试用例集的效能和覆盖范围。该过程主要包括三个关键步骤：生成变异、执行测试用例和分析结果。其中，生成变异的关键在于使用“变异算子”，这些算子是预定义的操作，如改变条件语句、替换运算符、修改变量值等，它们能系统性地引入可能的错误到源代码中。而这些基于一定的语法（Syntax）变换规则，通过对源程序进行程序变换（Program Transformation）得到的一系列变体，称之为变异体。对这些变异体执行测试用例，可以确定是否某个变异体导致了某个测试用例的失败，或者说测试用例是否“检测”到了该变异体。若测试用例能检测到变异体，该变异体被称作被“杀死”(Killed)；反之，则称为变异体“存活”(Survived)。通过变异得分计算公式，即变异得分 = 被杀死的变异体数/变异体总数，可以获取变异得分，进而对结果进行分析，以评估测试用例集的效能和覆盖范围。

为了更有效地实施变异测试，可以引入“变异测试工具”(Mutation Testing Tools) 的概念。这些工具是专门为自动化执行变异测试而设计的软件。这些工具可以通过多种方式集成到软件项目中，如通过可执行文件直接运行测试，或者与静态库、构建工具（如 Gradle 或 Maven）以及 IDE 插件配合使用。使用变异测试工具可以大大提高测试的效率和一致性，因为它们能够自动应用变异算子、执行测试用例并分析结果。

采用变异测试具有诸多显著的优点。首先，它能够覆盖大部分代码逻辑，不仅限于通过路径、分支或语句方式进行测试。其次，变异测试能够揭示测试用例集的不足之处，有助于识别那些未能检测出代码错误的测试。此外，由于变异测试鼓励开发更全面、更具代表性的测试用例，因此它可以提升软件的质量和可靠性。同时，通过对存活的变异体进行观察，开发者可以更深入地理解程序的行为和潜在的缺陷模式。

然而，变异测试也存在一些挑战和限制。主要的局限性包括高计算成本和可能的过度变异问题。由于生成和测试所有可能的变异体需要大量的计算资源和时间，这可能导致测试过程变得非常耗时。此外，过度变异可能会产生大量无关紧要或者等价的变异体，增加分析和处理的复杂性。

在实际应用中，变异测试对测试策略和质量保证流程的改进产生了深远影响。它推动了测试用例设计的精细化和自动化测试技术的发展，同时也促使开发者更加重视代码的可测试性和维护性。尽管存在挑战，但随着计算能力的提升和优化算法的进步，变异测试作为一种强大的软件质量保障手段，其价值和应用前景不容忽视。

2.2 发展历程

突变测试最早由 DeMillo、Lipton 和 Sayward 在 1978 年首次提出。

第一个实现突变测试工具的是 Tim A. Budd，他于 1980 年在他的博士论文中提出了一种针对 Fortan 语言的变异测试系统。

早期的变异测试应用可以追溯到 20 世纪 80 年代。Ntafos 是最早将变异测试作为测试集有效性的度量标准之一的研究者之一。Ntafos 将变异算子（例如常量替换）应用于 14 个 Fortran 程序的源代码中。生成的测试套件基于 3 种测试策略，即随机测试、分支测试和数据流测试，并针对变异分数进行评估。

DeMillo 和 Offutt 是第一个根据故障驱动的测试准则自动化生成测试数据的人。他们使用基于约束的测试（CBT）来自动化生成测试数据。

变异测试的另一个应用领域是调试，例如故障定位。在该领域比较早且较有影响力的研究是 Zhang 等人的于 2009 年发布的一篇文章，他们采用变异测试来研究巧合正确性对基于覆盖率的故障定位技术的影响。

3 综述准备

3.1 综述框架

对变异测试的背景和历史发展进行了了解后，我们明确定义了综述的研究范围，制定了以下综述协议以确保文献的有针对性和时效性：

(1) 聚焦于近 10 年（2010-2023）间的研究成果

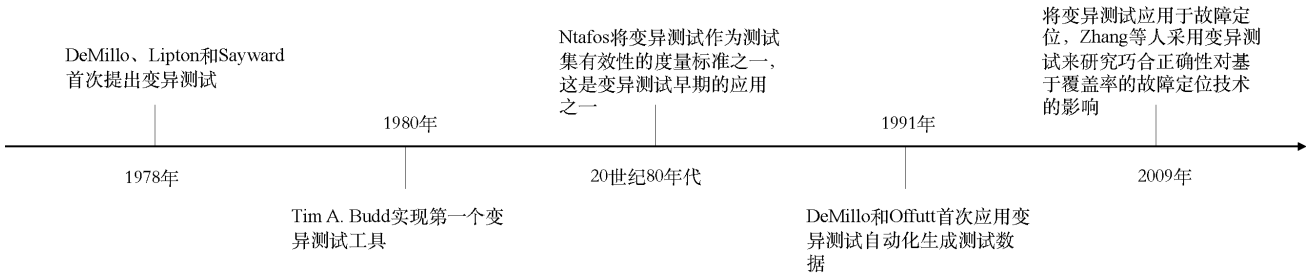

Fig. 1 发展历程

(2) 主要关注 CCFA、CCFB 列表中的会议和期刊

(3) 聚焦于变异测试在实际生产中的应用，而非变异测试的原理

(4) 变异测试在特点应用中包含优化工作是可以接受的

深入了解了变异测试的整体流程后，我们以导入待测程序 p 为起点，提出了 RQ1 和 RQ3，从生成变异算子出发提出了 RQ2，从未来研究角度提出 RQ4。我们通过对应用领域的分析，发现了变异测试在 AI 开发、测试中的应用已经成为趋势，故在 RQ3 中探索了变异测试应用在新旧趋势领域的不同。前人绝大多数文献都关注变异体的生成和质量的直接影响变异测试应用的有效性话题，所以我们在 RQ4 中详细探讨了变异测试面临的挑战等问题。以下是 RQ 详情:

(1) 近 10 年来变异测试在哪些领域应用较多?

(2) 近 10 年来变异测试在各个语言上的应用情况如何?

(3) 近年来变异测试如何应用于测试 AI，与传统软件测试有何区别?

(4) 近十年已有的研究还未解决哪些变异测试应用的问题? 或是他们的应用方法有何不足，需要解决?

3.2 文献检索与分类

在本研究的文献收集阶段，我们通过对 Google Scholar 和 DBPL 等学术数据库进行系统检索，以关键词 “Mutation Test” 为基础，收集了一系列与变异测试相关的文献。为确保文献的质量与相关性，我们依照 “综述协议” 对所收集到的文献进行了筛选。通过这一筛选机制，我们构建了一份文献库。这

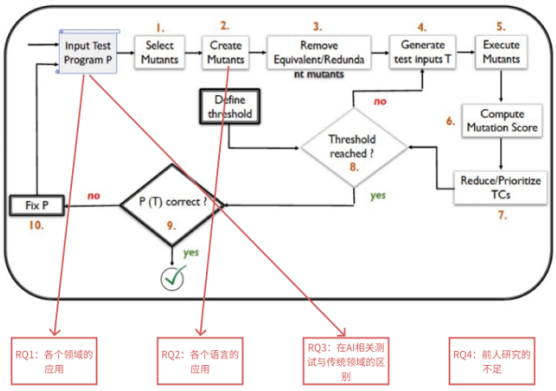

Fig. 2 综述框架

一文献库的形成为后续综述提供了坚实的基础，使得本论文能够在变异测试应用领域提供全面而准确的综述。

4 研究现状分析

4.1 RQ1 近 10 年来变异测试在哪些领域应用较多？

因变异测试的强功能和高价值，愈来愈多的领域开始应用变异测试。通过对所收集有关变异测试应用论文的统计分析，变异测试按照应用领域分类，大致可分为以下三类:

(1) 传统软件开发

(2) 智能软件、AI 开发

(3) 其他应用领域

其中传统软件开发约占 60%，智能软件、AI 开发约占 25%，其他应用领域约占 15%。

4.1.1 传统软件开发

在传统软件开发领域，变异测试多应用于嵌入式软件、Web 开发、Android 开发、数据库、建模

软件等领域。在这些领域中，变异测试可以作为一种软件质量保证和测试技术，展现出了广泛的实用价值和应用潜力。如在航天领域作为机载嵌入式软件的自动化测试评估方法 [1][2], 在 Android 开发中有 µDroid[3]、SE[4][5] 等变异测试工具保证软件安全性；变异测试也可以作为一种优化手段，如在数据库中提出 SchemaAnalyst 工具来自动检测和去除无效突变体 [6]，发现并修复数据一致性和性能相关的问题，提升数据库系统的整体效能。在 Web 开发领域，Shabnam Mirshokraie 等人提出了一种算法来选择变量和分支进行突变，避免了生成突变体后检测等效的突变体。[7]

4.1.2 智能软件、AI 开发

在深度神经网络 (Deep Neural Network, DNN) 领域，变异测试的应用多体现在模型验证 [8]、鲁棒性分析、优化探索等方面。为提升 DNN 的鲁棒性，Meixi Liu 等人提出了一种通用的数据导向变异框架——Styx，通过轻微变异训练数据来生成新的训练数据，这种方法能够在保持 DNN 在测试数据集上的准确性的同时，提高其对小幅度扰动的适应能力，从而提升鲁棒性 [9]。这一研究将变异测试的概念引入深度神经网络领域，通过创新的数据变异方法，有效地提高了 DNN 的稳健性和对小幅度扰动的抵抗能力。这不仅丰富了深度学习领域的测试技术和方法，也为保障 AI 系统的安全性和可靠性提供了新的思路和工具。

随着深度学习解决方案的广泛采用，如何有效地测试这些复杂系统成为了一个重大的研究挑战。Nargiz Humbatova 等人提出名为 DeepCrime 的变异工具，并遵循了系统化的程序，从现有的故障分类学中提取变异操作符，并在出现分歧时进行正式的冲突解决阶段，最终在 DeepCrime 中实现 24 个变异算子 [10]。这项研究为深度学习领域的变异测试提供了重要的理论基础和实践工具，有助于提高 DL 系统的可靠性和安全性，推动该领域测试技术的发展和应用。

4.1.3 其他应用领域

变异测试这一技术不仅在传统的软件开发领域中发挥了重要作用，而且正在逐渐扩展其应用范围，包括一些新兴的领域如教育科研和区块链技术。值得注意的是，在教育领域中，变异测试的应用是一个新颖且前所未有的发展趋势。

James Perretta 等人在评估学生编写的测试套件的质量时，引入了变异分析，自动在实现中插入潜在错误的方法，并证明变异分析产生的变异体是学生编写的代码中发现的错误的有效替代品，因而教师可以使用开源变异分析工具生成的变异体来评估学生的测试套件。以上发现对软件测试研究者、教育者和工具构建者都有影响 [11]。

因此，变异测试在教育领域的应用标志着一种创新的跨学科融合，它将软件工程的严谨性和教育科学的实践性相结合，为教育研究和实践带来了新的机遇和挑战。随着对变异测试在教育领域应用的深入探索和理解，我们有望看到更多的研究成果和实践案例，进一步推动教育领域的技术创新和教学质量的提升。

4.2 RQ2 近 10 年来变异测试在各个语言上的应用情况如何？

在调研变异测试在不同编程语言上的应用方面，我们对这些文献中涉及的编程语言进行了统计分析。统计结果表明，在变异测试的研究与应用中，Java 占据了总体的 0.43，C/C++ 占 0.23，Python 同样占 0.23，JavaScript 及其超集占据 0.19。此外，由于某些工具和研究适用于多种编程语言，因此这些比例之和可能大于 1。为了处理一些研究中未明确指出的语言以及包括了多语言适用性的情况，我们将一些语言如 PHP、C#、SQL 等归类为“其他”。

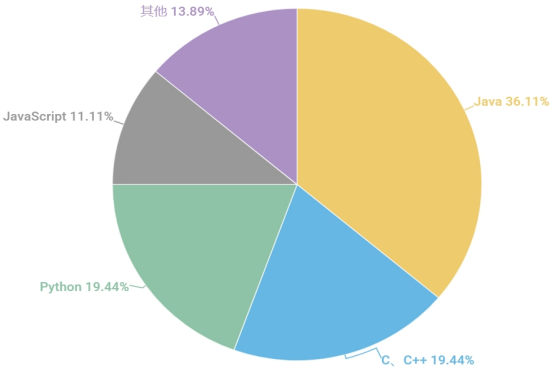

Fig. 3 综述文献语言占比

4.2.1 Java

在 Java 的应用方面，我们对收集的文献进行了多层次的研究，涵盖了字节码、抽象语法树 (AST)、源代码和安卓应用 APK 等不同层面，以进行突变算子的生成。在这些研究中，突变测试的实践不仅仅局限于代码层面，还包括了对字节码的变异、AST 的操作，以及针对安卓应用 APK 的突变测试。

在突变算子的生成方面，我们注意到一些研究对于突变算子的设计采用了多种角度。举例来说，MutAPK 2.0 首次提供了去除无用突变体和采样突变体集的功能，通过对安卓应用 APK 进行操作，为变异测试提供了更加灵活和实际的应用场景。[12]

在字节码层面的研究中，我们特别关注了 PIT（PITest）这一工具，由 Coles H 等人开发，目前是 GitHub 上使用最广泛的变异测试工具之一。PITest 通过扫描所有类，通过字节码操作生成突变体，并创建子 JVM 进程对每个突变体运行测试来进行评估。[13] 这种方法具有快速、易于使用的特点，能够在几分钟内完成分析，相较于以前的系统需要数天的时间来得更为高效。

此外，一些研究还尝试在生成变异算子时采用真实开发过程中的 bug，设计和提出测试框架。例如，Mohsen Moradi Moghadam 等人提出的 Akka 就是一个典型的例子。Akka 基于真实的开发缺陷，为突变测试框架的设计提供了更具实际应用背景的参考。[14]

4.2.2 C/C++

针对 C/C++，我们深入研究了在源代码层面进行突变算子生成的相关工作。Medina-Bulo 等人在这一领域应用了上下文搜索技术，通过遗传算法形成了 Evolutionary Mutation Testing。该方法从变异体子集出发，关注适应性高的个体变异体集合，以达到减少突变体数量、提高性能的目的。这种方法在突变体选择问题上的应用，通过结合上下文搜索技术和遗传算法，为 C/C++ 源代码的突变测试提供了一种创新而有效的解决方案。[15]

在 C/C++ 的突变测试工具集方面，SRCIROR 提供了在不同层面执行突变测试的方法。在 SRC（C / C++ 源代码）级别，SRCIROR 利用模式匹配技术在 Clang AST 上识别突变的程序结构。而在 LLVM 编译器中间表示（IR）级别，SRCIROR 通过改变 LLVM IR 指令来执行突变测试。这种多层次的突变测试方法使得 SRCIROR 能够全面而精确地评估 C/C++ 代码的质量和健壮性。[16]

值得一提的是，我们还收集到 Jaekwon Lee 等人采用模糊方法进行 C 和 C++ 软件的突变测试的研究。该方法以模糊的方法为基础，已被证明在 C 和 C++ 软件中是有效的。这为 C/C++ 领域的突变测试提供了另一种创新的方法。[17]

4.2.3 Solidity

值得提及的是，我们发现有一篇文献专注于 Solidity 语言的变异测试应用 [18]，Solidity 是一种用于智能合约开发的编程语言，特别设计用于以太坊区块链平台。该论文介绍的突变检测工具 MuSC 在 AST（抽象语法树）级别执行突变操作。MuSC 是第一个 ESC 的突变测试工具，它实现了一系列特定于最常用的 ESC 开发语言 Solidity 的突变算子 [18]。

4.2.4 Python

在对 Python 应用方面的调查中，引人注目的是我们发现所有相关论文均集中在 2017 年以后。这一时间节点标志着一个明显的转折，从而引发了我们对这种趋势的深入研究。在这个时间点之后，Java 应用的论文逐渐减少，而 Python 应用的数量则迅速增加，达到了 7 篇。我们推测，这可能与 2017 年人工智能领域取得了一系列重大突破有关，其中包括深度神经网络、自然语言处理中 Transformer 模型等关键技术的提出。这些技术突破推动了人工智能领域的迅速发展，而 Python 恰是大多数人工智能框架的首选语言。作为具体例证，深度神经网络的广泛应用是这一时期人工智能领域的一个显著特点。例如，深度学习中的图像识别、语音识别以及自然语言处理等任务取得了显著的进展，使得人工智能应用在各个领域都取得了令人瞩目的成就。Transformer 模型的提出则为自然语言处理任务引入了新的范 paradigm，取得了一系列在翻译、文本生成等方面的重大成功。

值得注意的是，尽管 Python 在人工智能领域占据主导地位，但变异测试在 Python 中的应用并非仅限于人工智能领域。以 Daniel Fortunato 在 2022 年提出的 QMutPy 工具为例 [19][20]，该工具被用于量子突变测试。类似于 Solidity，与 Solidity 相似，也是前人涉足较少的领域和语言，为变异测试提供了新的机遇。

4.2.5 小结

Ana B. Sanchez 等人的研究结果与我们的结论相一致，通过检索 GitHub 库中的变异测试工具，他们对各语言上的应用进行了分析。他们指出：“Infection（用于 PHP 的变异工具）是 GitHub 上目前使用最广泛的工具，约占找到的库的三分之一，其次是 PIT（Coles 等，2016）（用于 Java），Humbug（用于 PHP），StrykerJS（用于 JavaScript），以及 Mutant（用于 Ruby）。” [21]

但在我们的综述工作中 PHP 应用论文并未检索到较高占比。我们认为这可能涉及到两方面的原因：

首先，工业界和学术界的关注点可能存在差异，导致工业界应用与学术研究的同步性较低。这也是综述过程中可能面临的一种难题，需要在综述中准确反映工业实践和学术研究的最新动态，以确保全面而准确的综述结果。

其次，检索标准的过于严格可能导致了某些相关论文的遗漏。特别是对于 PHP 应用论文的检索，时间范围仅限于过去十年，并且局限在 CCF ABC 等标准，可能限制了对一些具有实际价值但不符合这些标准的论文的收集。在综述的进一步完善中，

可以考虑对检索标准进行适度放宽，以确保更全面地涵盖相关领域的研究成果。

变异测试在各语言应用值得关注方向总结：

(1) 变异算子的生成不仅局限于代码层面，还包括了对字节码的变异、抽象语法树（AST）的操作，以及针对安卓应用 APK 的突变测试。关注突变算子的多样性和灵活性，例如，通过去除无用突变体和采样突变体集提高实际应用性。

(2) 自 2017 年以后，Python 应用的突变测试论文数量迅速增加。但其并非局限于人工智能领域，还有量子计算领域等。

(3) 工业界应用与学术研究的同步性可能较低，是我们在综述过程中面临的挑战。

4.3 RQ3 近年来变异测试如何应用于测试 AI？与传统软件测试有何区别？

近年变异测试在 AI 领域的主要应用方向可以被概括为以下几个方面：

(1) 验证深度学习库所用测试用例的可靠度

(2) 在 AI 训练中应用基于突变的故障定位

(3) 对 AI 训练数据进行突变

(4) 针对 AI 的变异测试框架

4.3.1 验证深度学习库所用测试用例的可靠度

上海交通大学 Li Jia[22] 等人以 TensorFlow, Theano and Keras 三个广泛使用的深度学习库为例，将变异测试应用于深度学习库上，以验证其测试用例的可靠性。该团队使用 13 种突变算子对深度学习库代码进行突变，找寻被杀死的变异体，并定位是哪些测试用例杀死了变异体。他们使用变异测试方法发现了深度学习库测试用例在质量上的缺陷，并总结了“现有的单元测试用例更有效地检测由错误逻辑流程引起的错误，而不是由错误数值引起的错误。”等 11 条发现。

4.3.2 在 AI 训练中应用基于突变的故障定位

Ali Ghanbari[8] 等人提出了 deepmufl，一种适用于 DNN 的基于突变的故障定位技术。该技术能快速帮助开发人员定位模型训练代码中的错误。该技术基于对预训练的 DNN 模型进行突变的思想，并根据 Metallaxis 和 MUSE 方法、Ochiai 和 SBI 公式以及突变对测试数据点结果的两种影响类型来计算可疑性值，以定位可能的错误位置。该团队进行的包含 109 个模型错误的基准测试指出，与其他最先进的静态和动态故障定位系统相比，尽管 deepmufl 比其他工具慢，但它在检测到的总错误数量方面几乎是它们的两倍，它检测到了其中没有一个研究工具能够检测到的 21 个错误。

4.3.3 对 AI 训练数据进行突变

近年来在 AI 领域提出的数据导向突变框架包括国防科技大学 Meixi Liu[9] 等人提出的通用的数据导向突变框架 Styx，以及 Guandi Liu[23] 等人提出的点云数据突变工具 TauPad。

Styx 是一个通用的数据导向突变框架，可以提高 DNN 的健壮性。使用 Styx 可以由原数据集生成一个新的训练数据集，该数据集通过轻微突变原始数据而产生。实验结果表明，Styx 在测试准确性与传统训练方法相似的前提下，提高了训练模型的健壮性。

点云在许多应用场景中被广泛用于处理各种深度学习（DL）任务。而 TauPad 是一个针对点云这一数据类型的突变工具。该工具使用对抗性攻击引导突变方向。基于点云预处理、点云对抗性突变和空间分布恢复，TauPad 能够生成对目标模型具有显著欺骗性的增强测试数据。初步实验表明，TauPad 可以可靠且有效地增强用于测试的点云。相比其他点云变异技术，它可以增强测试集的点云，而不违反其空间结构。

4.3.4 针对 AI 的变异测试框架

近年提出的 DL 系统突变测试框架主要有 DeepMutation++ 和 DeepCrime。

DeepMutation++[24] 是一种基于突变测试的 DNN 工具，它促进了 DNN 的质量评估，支持前馈神经网络（FNNs）和有状态循环神经网络（RNNs）。不仅能够对整体输入对 DNN 模型的健壮性进行静态分析，而且还允许通过运行时分析识别顺序输入（例如音频输入）的脆弱部分。在给定要分析的 DNN 模型和一组测试数据的情况下，DeepMutation++ 首先利用提供的突变算子生成一组高质量的 DNN 变异体。在生成了一定数量的变异体之后，DeepMutation++ 分析原始 DNN 与生成的 DNN 变异体在对抗提供的测试输入时的行为差异，进行健壮性分析。最后，DeepMutation++ 输出指示测试数据质量和 DNN 健壮性的分析报告。

DEEPCRIME[25][10] 是一个基于真实故障的 DL 系统突变测试工具，模拟真实 DL 故障发生以进行 DL 系统的突变测试。其团队提出并实现了 24 个突变算子，这些算子是系统地从现有的关于真实 DL 故障的研究中提取出来的。作为输入，该工具接收 DL 系统的源代码、训练和测试数据集以及所有必要的依赖关系。然后，它通过为要应用的每个突变算子引入故障来生成突变体。存储原始和变异的训练模型在训练和测试数据集上的性能。然后，对

于创建的每个变异体，确定它是否被杀死。最后，有一个文件保存了每个应用的突变算子的测试集的突变分数以及通过所有运算符计算的总突变分数的平均值。相比 DeepMutation++，DEEPCRIME 在区分测试集的质量方面更有效。

4.3.5 变异测试在 AI 领域与在传统领域应用的区别

尽管在深度学习系统中使用突变测试是一种十分有应用前景的质量保证技术，但杀死、等效变异体这些熟悉的概念并不能直接应用在深度学习系统中。在传统软件系统中，如果原始程序和变异体在测试场景中表现不同，则认为变异体被杀死。在深度学习系统中，即使在相同数据集上重新训练系统而不对其进行任何变异，通常也会观察到行为差异，这是因为训练过程通常是非确定性的。因此，对于单个测试输入的不良行为是不足以杀死突变体的。[26]

一些团队使用了其他标准来定义变异体是否被杀死。例如，DEEPCRIME[10] 研究团队对原始模型和突变模型进行 n 次训练，统计比较原始模型和变异体的性能测量分布（例如准确度或损失），如果性能测量分布的差异在被判断为具有统计显著性的情况下是非小且非可忽略的，则认为突变体被杀死。

4.4 RQ4 **近十年已有的研究还未解决哪些变异测试应用的问题？或是他们的解决方法有何不足，需要解决？**

在变异测试应用领域的近十年已有研究中，大量工具与框架被开发用在各个语言和领域当中，而伴随着新兴复杂语言与领域的出现，变异测试的应用研究也不断面临着新的问题与不足。通过对这些研究的分析，本文将这些问题归纳为以下三个方面：

(1) 变异体的生成和质量

(2) 大规模软件系统的可扩展性

(3) 整合到软件开发流程的挑战

4.4.1 变异体的生成和质量

变异体的生成和质量一直是变异测试的核心关注点，直接影响变异测试的准确性和覆盖范围。应对不同的语言和领域，要想充分发挥变异测试的作用，就绕不开变异体的生成和质量的这个话题。该话题可以划分为以下几个方面：

(1) 多样性和有效性的平衡

(2) 处理复杂语言特性和代码结构

(3) 变异体与真实缺陷的关联性

1. 多样性和有效性的平衡：变异测试需要生成足够多样的变异体，以确保测试用例能够覆盖潜在的缺陷。然而，在追求多样性的过程中，可能会引入非生产或不切实际的变异，从而降低了测试的有效性。在一篇提出了 PIT——用于 Java 的突变测试工具的论文 [13] 中有如下的详细叙述：

”PIT supports a small number of mutation operators, the objective being to limit the number of mutants and the execution time, but with the risk to have a set of generated mutants of low quality.We have recently proposed to extend the list of mutation operators to increase the effectiveness of the tool. ”

这充分说明了 PIT 研究人员为了限制变异体的数量和执行时间而选择采用少量的变异算子，但是会产生低质量突变体的问题，即忽略了多样性而关注变异体的约减会导致覆盖率不足；通过扩展突变操作符列表（追求多样性）能够适度地提升有效性。然而，在追求多样性的过程中，可能会引入非生产或不切实际的变异，从而降低了测试的有效性。例如，鉴于突变体冗余与等价的多样性问题，Medina-Bulo[15] 等人将上下文搜索技术应用于突变体选择问题，关注适应性高的个体变异体集合来减少突变体的数量，提高性能。

2. 处理复杂语言特性和代码结构：处理复杂语言特性和代码结构是变异测试领域的一个重要问题，尤其在现代编程语言中，包含复杂的语法和丰富的语言特性，这些往往需要变异测试的针对性开发才能有效地集成应用。最近研究中研究了比如 MATLAB，JVM 字节码语言，Solidity，Haskell 等复杂语言。具体如 Haskell，一种函数式编程语言，Duc Le 等学者提出了 MuCheck[27]，一个用于 Haskell 程序的变异测试工具，该工具实现了专门为函数式程序设计的突变操作符，并利用 Haskell 的类型系统来实现比其他方法更相关的突变集。由于高度紧凑的代码、引用透明性和清晰的语义，变异覆盖对函数式程序特别有价值；这使得基于存活的变异体增加测试套件或规范成为改进测试的实用方法。

3. 变异体与真实缺陷的关联性：解决这些问题的方法包括改进变异操作的设计，引入更多的复杂性，结合其他测试方法，以及通过实证研究不断改进变异测试的技术和实践。这就引出了变异测试优化另一个大的研究领域。在使用变异测试时，测试人员应意识到这些限制，并在结果解释和测试计划中予以考虑。在 Mike Papadakis[28] 的一篇相关论文也提到：最近的研究对故障与变异关系的某些方面

进行了研究，如变异死亡与真实故障检测之间的关系和变异测试的故障检测能力。Just 等人报道“变异体检测与真实故障检测之间存在统计学上显著的相关性，与代码覆盖率无关”，而 Chekam 等人则认为“只有在达到相对较高的覆盖率时，故障发现才会开始显著增加”。这说明变异体与真实缺陷的关联性还存在着争议与讨论，也是变异测试应用研究的一个待考量的方面。

4.4.2 大规模软件系统的可扩展性

考虑到变异测试能够有效实现软件测试活动评估的自动化，变异分析已被证明是测试套件自动评估的一个有前途的解决方案，它包括根据导致测试失败的注入错误的百分比来度量测试套件的质量。然而变异测试可扩展性的优化问题仍然需要改进以促进变异测试的工业应用。

在 Mutation Analysis for Cyber-Physical Systems: Scalable Solutions and Results in the Space Domain[1] 中也提到：到目前为止，两个主要问题阻碍了空间机构在空间软件开发中实施变异分析。首先，在这种情况下应用变异分析优化技术的可行性存在不确定性。其次，大多数现有技术要么会打破嵌入式软件中常见的实时性要求，要么无法在软件验证设施 (包括 CPU 模拟器和传感器模拟器) 中对软件进行测试。以上论文增强了突变分析优化技术，使其适用于嵌入式软件，并提出了一个成功集成它们的管道，以解决上述背景下的可扩展性和准确性问题。

但是上述研究依然存在着可扩展性方面的不足：不能评估测试套件是否能够检测到与异构组件之间的通信或不同技术和工具的互操作性有关的故障。因此，要想变异测试集成到大规模软件系统中，依然存在着许多限制与挑战。

4.4.3 整合到软件开发流程的挑战

变异测试应用于软件开发流程也是近年研究的热门话题，变异测试能够检测测试套件的缺陷，帮助开发人员更好地理解其测试用例的覆盖范围。同时，集成变异测试到持续集成流程中，使得开发人员能够在每次提交代码后获得快速的测试反馈。这有助于及早发现和解决问题，促使敏捷开发和快速迭代。随着软件开发流程的迭代发展，将变异测试整合到软件开发流程也同时面临着新的挑战，大致可分为以下四点：

(1) 工具集成和适配挑战

(2) 变异测试与其他测试方法的融合

(3) 变异测试结果解释和过滤

工具集成和适配挑战：将变异测试工具集成到已有的开发环境和工具链中，确保与其他测试工具和系统的兼容性可能是一项复杂的任务。比如”PIT: A Practical Mutation Testing Tool for Java” 论文 [13] 为了解决这方面的问题，介绍了 PIT（Pitest），这是一个针对 Java 项目的实用变异测试工具。PIT 被设计成易于配置，并能够嵌入到常规的构建过程中，它也是健壮的，并且与开发工具集成得很好，因为它可以通过命令行接口，Ant 或 Maven 来调用。

变异测试与其他测试方法的融合：如何将变异测试与其他测试方法如静态分析、动态分析等融合，以提高测试的全面性和效果也是变异测试应用需要解决的问题。Anna Derezińska 在一篇论文 [29] 中实现了类似的技术：在测试驱动的开发中，基本测试是在程序编码之前为其准备的。许多短的开发周期在过程中重复，需要对准备好的测试和测试代码提取进行快速响应。用于评估和开发测试用例的突变测试，通常需要花费相当长的时间来获得有价值的测试评估，在 VisualMutator 中实现了这些技术的融合。

变异测试结果解释和过滤：变异测试结果解释和过滤的重要性在于提高测试效率、减少误报，以及更好地理解测试覆盖和代码质量，也是应用领域重点关注的挑战之一。在 practical Mutation Testing at Scale: A view from Google 一文 [30] 中，Google 提出了一种可扩展的突变测试方法:(1) 突变测试是增量的，在代码审查期间仅对更改的代码进行突变，而不是对整个代码库进行突变;(2) 对突变体进行过滤，去除可能与开发人员无关的突变体，并限制每行和每个代码审查过程的突变体数量;(3) 根据突变算子的历史表现选择突变体，进一步剔除无关突变体，提高突变体质量。然而以上提出的方案在结果解释和过滤上还有需要改进的问题：没有考虑到非冗余突变体可能是非生产性的，或者等效突变体可能是生产性的。

5 结论

本文对变异测试的研究现状和潜在发展方向进行了分析。我们不仅发现了前人在应用变异测试时创新性的工作如利用遗传算法生成变异算子，还重点分析了 AI 测试作为新兴趋势研究领域，变异测试在其中的应用细节以及与其他领域的区别。最终我们总结了文献中中存在的局限性和尚待解决的问题，如变异体生成的质量等，以启示未来的研究方向。我们希望综述工作能够展示这些方向，促进变异测试的更广泛、更有效的应用。

参 考 文 献

[1] O. Cornejo, F. Pastore, and L. C. Briand, “Mutation analysis for cyber-physical systems: Scalable solutions and results in the space domain,” IEEE Transactions on Software Engineering, vol. 48, no. 10, pp. 3913–3939, 2021.

[2] O. Cornejo, F. Pastore, and L. Briand, “Mass: A tool for mutation analysis of space cps,” in Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, 2022, pp. 134–138.

[3] R. Jabbarvand and S. Malek, “μdroid: an energy-aware mutation testing framework for android,” in Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 208–219.

[4] A. S. Ami, K. Kafle, K. Moran, A. Nadkarni, and D. Poshyvanyk, “Systematic mutation-based evaluation of the soundness of security-focused android static analysis techniques,” ACM Transactions on Privacy and Security (TOPS), vol. 24, no. 3, pp. 1–37, 2021.

[5] R. Bonett, K. Kafle, K. Moran, A. Nadkarni, and D. Poshyvanyk, “Discovering flaws in {Security-Focused} static analysis tools for android using systematic mutation,” in 27th USENIX Security Symposium (USENIX Security 18), 2018, pp. 1263–1280.

[6] P. McMinn, C. J. Wright, C. J. McCurdy, and G. M. Kapfhammer, “Automatic detection and removal of ineffective mutants for the mutation analysis of relational database schemas,” IEEE Transactions on Software Engineering, vol. 45, no. 5, pp. 427–463, 2017.

[7] S. Mirshokraie, A. Mesbah, and K. Pattabiraman, “Guided mutation testing for javascript web applications,” IEEE Transactions on Software Engineering, vol. 41, no. 5, pp. 429–444, 2014.

[8] A. Ghanbari, D.-G. Thomas, M. A. Arshad, and H. Rajan, “Mutation-based fault localization of deep neural networks,” in 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2023, pp. 1301–1313.

[9] M. Liu, W. Hong, W. Pan, C. Feng, Z. Chen, and J. Wang, “Styx: A data-oriented mutation framework to improve the robustness of dnn,” in Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, 2020, pp. 1260–1261.

[10] N. Humbatova, G. Jahangirova, and P. Tonella, “Deepcrime: from real faults to mutation testing tool for deep learning,” in 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). IEEE, 2023, pp. 68–72.

[11] J. Perretta, A. DeOrio, A. Guha, and J. Bell, “On the use of mutation analysis for evaluating student test suite quality,” in Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 2022, pp. 263–275.

[12] C. Escobar-Velásquez, D. Riveros, and M. Linares-Vásquez, “Mutapk 2.0: A tool for reducing mutation testing effort of android apps,” in Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 1611–1615.

[13] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, “Pit: a practical mutation testing tool for java,” in Proceedings of the 25th international symposium on software testing and analysis, 2016, pp. 449–452.

[14] M. Moradi Moghadam, M. Bagherzadeh, R. Khatchadourian, and H. Bagheri, “ akka: Mutation testing for actor concurrency in akka using real-world bugs,” in Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2023, pp. 262–274.

[15] P. Delgado-Pérez and I. Medina-Bulo, “Search-based mutant selection for efficient test suite improvement: Evaluation and results,” Information and Software Technology, vol. 104, pp. 130–143, 2018.

[16] F. Hariri and A. Shi, “Srciror: A toolset for mutation testing of c source code and llvm intermediate representation,” in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 860–863.

[17] J. Lee, E. Viganò, O. Cornejo, F. Pastore, and L. Briand, “Fuzzing for cps mutation testing,” in 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2023, pp. 1377–1389.

[18] Z. Li, H. Wu, J. Xu, X. Wang, L. Zhang, and Z. Chen, “Musc: A tool for mutation testing of ethereum smart contract,” in 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019, pp. 1198–1201.

[19] D. Fortunato, J. Campos, and R. Abreu, “Qmutpy: A mutation testing tool for quantum algorithms and applications in qiskit,” in Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 2022, pp. 797–800.

[20] ——, “Mutation testing of quantum programs written in qiskit,” in Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, 2022, pp. 358–359.

[21] A. B. Sánchez, P. Delgado-Pérez, I. Medina-Bulo, and S. Segura, “Mutation testing in the wild: findings from github,” Empirical Software Engineering, vol. 27, no. 6, p. 132, 2022.

[22] L. Jia, H. Zhong, and L. Huang, “The unit test quality of deep learning libraries: A mutation analysis,” in 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2021, pp. 47–57.

[23] G. Liu, J. Liu, Q. Zhang, C. Fang, and X. Zhang, “Taupad: test data augmentation of point clouds by adversarial mutation,” in Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, 2022, pp. 212–216.

[24] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu, and J. Zhao, “Deepmutation++: A mutation testing framework for deep learning systems,” in 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019, pp. 1158–1161.

[25] N. Humbatova, G. Jahangirova, and P. Tonella, “Deepcrime: mutation testing of deep learning systems based on real faults,” in Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 67–78.

[26] G. Jahangirova and P. Tonella, "An empirical evaluation of mutation operators for deep learning systems," in 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST). IEEE, 2020, pp. 74–84.

[27] D. Le, M. A. Alipour, R. Gopinath, and A. Groce, "Mucheck: An extensible tool for mutation testing of haskell programs," in Proceedings of the 2014 international symposium on software testing and analysis, 2014, pp. 429–432.

[28] M. Papadakis, D. Shin, S. Yoo, and D.-H. Bae, "Are mutation scores correlated with real fault detection? a large scale empirical study on the relationship between mutants and real faults," in Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 537–548.

[29] A. Derezińska and P. Trzpil, "Mutation testing process combined with test-driven development in. net environment," in Theory and Engineering of Complex Systems and Dependability: Proceedings of the Tenth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, June 29–July 3 2015, Brunów, Poland. Springer, 2015, pp. 131–140.

[30] G. Petrović, M. Ivanković, G. Fraser, and R. Just, "Practical mutation testing at scale: A view from google," IEEE Transactions on Software Engineering, vol. 48, no. 10, pp. 3900–3912, 2021.