软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(1):62−101 [doi: 10.13328/j.cnki.jos.007450] [CSTR: 32375.14.jos.007450]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向智能软件系统的测试用例生成方法综述*

吉 品, 冯 洋, 吴 朵, 刘 嘉, 赵志宏

(计算机软件新技术全国重点实验室 (南京大学), 江苏 南京 210046)

通信作者: 冯洋, E-mail: fengyang@nju.edu.cn; 刘嘉, Email: liujia@nju.edu.cn

摘 要: 随着深度学习等技术的快速发展以及计算机硬件、云计算等领域的重大突破, 日益成熟的人工智能技术已经被应用于不同场景的软件系统中. 这类以人工智能模型为核心组件的软件系统, 统称为智能软件系统, 按照人工智能技术的应用领域可分为图像处理应用、自然语言处理应用、语音处理应用等. 与传统软件系统不同, 人工智能模型采用数据驱动的编程范式, 其中所有的决策逻辑均通过大规模数据集学习得到. 这种范式的转变导致传统的基于代码的测试用例生成方法无法用于智能软件系统的质量评估. 因此, 近年来许多研究人员致力于面向智能软件系统的测试方法研究, 包括结合智能软件系统的特点提出新的测试用例生成方法、测试用例评估方法等. 围绕面向智能软件系统的测试用例生成方法调研 80 篇相关领域的文献, 将现有方法按照适配的系统类型进行分类, 对面向图像处理应用、自然语言处理应用、语音处理应用、点云处理应用、多模态数据处理应用以及深度学习模型的已有测试用例生成方法进行系统地梳理和总结. 最后, 对面向智能软件系统的测试用例生成方法的未来工作进行展望, 以期为该领域的研究人员提供参考.

关键词: 人工智能软件系统; 软件测试; 测试用例生成; 深度学习

中图法分类号: TP311

中文引用格式: 吉品, 冯洋, 吴朵, 刘嘉, 赵志宏. 面向智能软件系统的测试用例生成方法综述. 软件学报, 2026, 37(1): 62–101. http://www.jos.org.cn/1000-9825/7450.htm

英文引用格式: Ji P, Feng Y, Wu D, Liu J, Zhao ZH. Survey on Test Case Generation Methods for Intelligence Software Systems. Ruan Jian Xue Bao/Journal of Software, 2026, 37(1): 62–101 (in Chinese). http://www.jos.org.cn/1000-9825/7450.htm

Survey on Test Case Generation Methods for Intelligence Software Systems

JI Pin, FENG Yang, WU Duo, LIU Jia, ZHAO Zhi-Hong

(State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210046, China)

Abstract: With the rapid development of technologies such as deep learning and significant breakthroughs in areas including computer hardware and cloud computing, increasingly mature artificial intelligence (AI) technologies are being applied to software systems across various fields. Software systems that incorporate AI models as core components are collectively referred to as intelligence software systems. Based on the application fields of AI technologies, these systems are categorized into image processing, natural language processing, speech processing, and other applications. Unlike traditional software systems, AI models adopt a data-driven programming paradigm in which all decision logic is learned from large-scale datasets. This paradigm shift renders traditional code-based test case generation methods ineffective for evaluating the quality of intelligence software systems. As a result, numerous testing methods tailored for intelligence software systems have been proposed in recent years, including novel approaches for test case generation and evaluation that address the unique characteristics of such systems. This study reviews 80 relevant publications, classifies existing methods according to the types of systems they target, and systematically summarizes test case generation methods for image processing, natural language processing, speech processing, point cloud processing, multimodal data processing, and deep learning models. Potential future directions for test case generation in intelligence software systems are also discussed to provide a reference for researchers in this field.

* 基金项目: 国家自然科学基金 (62372225, 62272220)
收稿时间: 2023-11-07; 修改时间: 2025-01-17, 2025-03-25; 采用时间: 2025-04-23; jos 在线出版时间: 2025-09-03
CNKI 网络首发时间: 2025-09-04

Key words: artificial intelligence software system; software testing; test case generation; deep learning

近年来, 人工智能 (artificial intelligence, AI) 技术发展迅猛, 在多个研究领域取得了重要突破, 常见的人工智能应用包括语音识别及理解 (如 Siri[1] 和 Alexa[2]), 自然语言理解及生成 (如 ChatGPT[3] 和 ERNIE Bot[4]), 自动驾驶汽车 (如 Waymo[5] 和 Cruise[6]) 等. 人工智能技术的普及已经改变了人们的生活方式[7]. 与此同时, 核心组件为人工智能模型的智能软件系统的质量问题也受到关注. 智能软件与传统类型的软件一样, 会存在软件缺陷. 这些缺陷有时会带来严重后果, 轻则是经济损失[8], 重则可能引发人身安全威胁[9]和政治纠纷[10]. 因此, 对智能软件系统进行充分和完备的测试至关重要, 我们迫切需要找到潜在缺陷以准确地评估这类系统在真实应用场景中的性能, 为进一步提升智能软件的质量提供数据基础和迭代改进思路.

当前, 标准的人工智能模型评估范式是使用基准数据集中的验证集或者测试集来获取表示模型性能的一些指标, 比如准确率 (Accuracy), 精确率 (Precision) 和召回率 (Recall) 等[11]. 然而, 基准数据集中的验证集和测试集有着和训练集相同的数据偏差, 这会导致被测对象在真实应用场景中的性能被高估[12,13]. 因此, 使用标准的评估范式获取到的评估结果并不精确, 但高昂的收集及处理成本又使得高质量的测试数据非常稀缺. 并且, 使用聚合的统计结果来反映模型或系统的性能, 不利于找出缺陷的具体表现形式, 难以继续探索对应的修复方式[14]. 除此之外, 现有研究表明, 使用基准数据集进行评估并不能真实地反映出系统的各项能力水平, 无法分辨出模型开发者是否使用了一些“技巧”来提升模型的性能[15]. 比如, 用户常要求自然语言处理应用拥有较强的词语辨析、语义理解等能力. 如果仅依据上述指标, 测试人员无法判断出机器阅读理解、智能问答等应用程序是否使用了句式匹配、关键词记忆等“技巧”去回答用户的问题, 因此也就无法明确被测对象是否真正达到了预设目标.

为了解决当前存在的问题, 研究人员已经聚焦于智能软件系统的测试方法的研究. 与传统软件不同, 智能软件系统中的所有决策逻辑都是从大规模数据集学习得到, 具有内部结构复杂, 参数规模巨大等特点, 难以使用传统的软件测试技术进行测试. 目前, 研究人员已经提出了多种面向智能软件系统的测试用例生成方法, 这些方法能够通过使用合适的生成策略产生多样化的测试用例, 确保能够覆盖包括边界条件的不同测试场景. 同时, 为保证测试用例的有效性与结果的可靠性, 还需在多样化生成的基础上注重维护测试用例的一致性和准确性. 根据测试用例的执行结果, 测试人员能够进一步分析得出智能软件系统在正确性、鲁棒性、安全性和公平性等多个维度的表现情况. 因此, 遵循数据驱动的测试思路, 探究如何使用足够且适配的测试用例对智能软件系统的一般行为和边界条件下的行为进行评估是必要和可行的. 在测试用例生成过程中, 不同类型的测试输入对测试效果具有重要影响. 智能软件系统的测试输入可分为对抗输入和自然输入, 其中对抗输入是在原始输入中增加扰动, 可能偏离正常数据分布; 而自然输入指属于实际应用场景数据分布的输入[16]. 本文重点关注基于应用领域特征生成自然输入测试用例的相关工作.

目前, 已有多篇文献从不同的角度对现有的机器学习 (machine learning, ML) 以及深度学习 (deep learning, DL) 测试领域的相关研究成果进行了综述. Xiang 等人[17]总结了用于机器学习、自治系统以及神经网络的形式化验证方法, 并重点关注如何确保信息物理融合系统的安全性. Huang 等人[18]从验证、测试、对抗攻击和可解释性这 4 个方面, 对深度神经网络 (deep neural network, DNN) 的安全性和可靠性相关的研究工作进行了综述. Zhang 等人[16]梳理了机器学习测试中的测试属性、测试组件、测试流程以及应用场景, 并进一步分析了数据集特点, 总结了研究趋势. Braiek 等人[19]从发现的错误类型对机器学习测试方法进行了分类总结, 包括检测数据质量、检测机器学习模型概念化错误以及模型实现错误的方法. 王赞等人[20]从深度神经网络测试度量指标、测试输入生成、测试预言方面对已有深度神经网络测试的研究成果进行了系统梳理.

虽然上述综述从不同角度总结了相关测试领域的工作, 但大多侧重于测试方法的技术属性或通用性讨论, 未能充分体现不同应用领域中的独特测试需求和挑战. 特别是对于智能软件系统, 由于应用场景、任务目标及输入数据特征的差异, 不同领域的系统往往需要定制化的测试用例生成方法. 为此, 本文从应用领域角度出发, 结合系统任务目标, 梳理与归纳以人工智能模型为核心、数据模态多样、应用场景各异的智能软件系统的测试用例生成方法, 相较于以往综述更突出领域差异性与针对性. 本文充分调研了智能软件测试领域的相关研究成果, 重点关注

结合智能软件系统任务目标特征与应用场景特征时, 如何为各类智能软件系统生成真实有效的测试输入, 如何解决测试预言缺失问题, 以及如何引导测试用例生成以提高测试效率. 此外, 本文区别于以往泛化面向机器学习或深度学习模型的测试流程, 强调领域特异性问题的解决. 具体来说, 本文首先从应用领域以及完成的任务目标对智能软件系统进行分类, 再对面向各类智能软件系统的测试用例生成方法进行梳理和总结, 最后对该研究领域的未来提出展望, 以此帮助研究人员系统了解不同类型智能软件系统测试用例生成方法的研究现状及发展趋势.

本文第 1 节从智能软件系统的概念定义、智能软件系统测试存在的挑战等方面系统介绍背景知识. 第 2 节介绍本文的文献收集、筛选以及汇总的过程. 第 3–7 节分别从图像处理应用、自然语言处理应用、语音处理应用、点云处理应用以及多模态数据处理应用这 5 个软件类型的角度详细地梳理总结面向不同类型智能软件系统的测试用例生成方法. 第 8 节对通用性的面向深度学习模型的测试用例评估指标和测试用例生成方法进行汇总整理. 第 9 节对智能软件测试用例生成未来的研究方向以及挑战进行展望与分析. 第 10 节对本文的内容进行总结.

1 背　景

本节主要介绍智能软件系统的概念定义、采用的关键技术和对应的测试难点以及面向智能软件系统的测试用例生成的研究背景与框架.

1.1 智能软件系统

随着深度学习等技术的快速发展以及计算机硬件、云计算等领域的重大突破, 人工智能技术日益成熟, 已广泛应用于不同场景的软件系统中. 人工智能通常是指数字计算机或计算机控制的机器人能够自主地或交互地执行拟人任务[21], 其相关科学研究领域包括机器学习 (ML)、自然语言处理 (natural language processing, NLP)、计算机视觉 (computer vision, CV)、语音识别与合成 (speech recognition and synthesis, SRS)、机器人和专家系统等[22]. 基于人工智能技术的应用软件已经与人们的日常生活密不可分, 包括机器翻译[23]、语音识别[24]和对话机器人[25]等. 此外, 人工智能技术也被应用于一些安全关键或者任务关键的生命攸关系统, 如自动驾驶[26]、手术机器人[27]等. 本文将这些采用了人工智能技术, 集成了人工智能模型, 能够模仿或再现人类行为和思维, 实现学习、推理和解决问题的系统, 统称为智能软件系统[28].

目前, 迅猛发展的深度学习技术逐渐成为智能软件系统所运用的关键技术[20]. 深度学习最早由 Hinton 等人[29]在 2006 年提出, 指一种基于样本数据, 通过特定训练方法获得多层次网络结构的机器学习过程. 其动机在于建立模拟人类大脑机制的深度神经网络, 用于进行数据学习和分析. 深度神经网络是由相互连接的计算单元层构成, 就像大脑中的神经元以及之间的连接一样. 深度神经网络极大地提高了语音识别、计算机视觉、自然语言处理等领域的先进水平[30]. 比如, 卷积神经网络 (convolutional neural network, CNN) 提取出的特征会更多地关注局部, 参数共享的特点又能够很大限度地减少运算量, 因此常被用于图像处理应用[31]; 循环神经网络 (recurrent neural network, RNN) 具有记忆性, 因此常被用于时序性的特征学习, 在语音识别、机器翻译等领域常被应用[32]; 以 RNN 为基础的长短期记忆网络 (long short-term memory, LSTM) 可以保留长序列数据中的重要信息和忽略不重要信息, 解决了 RNN 的梯度消失和爆炸问题, 给各个应用场景带来了进一步的效果提升[33].

从上文可以看出, 以深度学习模型为代表的人工智能模型的性能对于智能软件系统的质量来说至关重要, 因此人工智能模型常被定义为智能软件系统中的核心部件. 与内部逻辑由软件开发人员手动编写代码确定的传统软件系统不同, 人工智能模型通常采用数据驱动的编程范式, 其中所有的决策逻辑都是从大规模数据集中学习而得[34–36]. 这种“端到端”决策模式导致模型的可解释性极低, 结构复杂、参数众多的模型常被视为一个“黑盒”. 由于这种范式的转变, 基于代码的传统软件测试方法已无法对智能软件系统进行全方位的质量评估. 智能软件系统与传统软件系统一样, 也存在缺陷, 且对于生命攸关系统来说, 缺陷可能导致严重后果. 例如, 2016 年和 2020 年, 处于自动驾驶模式下的特斯拉汽车均发生了未减速撞击前方白色卡车并导致人员伤亡的交通事故[37]. 因此, 智能软件系统的质量问题吸引了工业界以及学界的大量关注, 捕获智能软件系统的缺陷、评估其在真实应用场景中的性能, 已成为当前研究的重点方向.

1.2 面向智能软件系统的测试用例生成

智能软件系统具有处理数据模态多样、内部结构复杂、应用场景多变的特性, 这导致在测试过程中难以模拟出接近真实应用场景中的测试输入和构建出正确全面的测试预言. 若依赖人工从真实应用场景收集测试数据, 随着测试需求的增加, 将面临人工标注成本高昂和测试场景覆盖率难以控制的问题. 目前, 已经有许多研究人员从智能软件系统完成的任务目标及输入数据的特性入手, 提出了多种面向不同应用领域的测试用例生成方法以解决上述挑战[16]. 通常来说, 针对智能软件系统的测试用例生成方法都包含测试输入生成和测试预言生成这两个步骤. 为了提升测试效率, 有的研究者会在测试用例生成过程中使用引导算法, 以提高生成的测试用例质量. 测试输入生成和测试预言生成方法都需要基于被测智能软件系统的所在应用领域、完成任务以及输入数据的特征进行设计, 输入数据和测试预言共同构成一个完整的测试用例.

图 1 描述了当前面向智能软件系统的测试用例生成方法的研究框架. 在构造测试输入时, 模糊测试方法常被用于自动化生成测试输入, 通过对种子输入数据实施随机变换或者按照特定规则变换, 来获取到多个新输入, 以缓解测试数据获取困难的问题[38]. 除此之外, 变异测试、符号执行等方法都被用于生成输入数据. 在测试预言生成中, 研究者们通常采取蜕变测试和差分测试方法, 以应对正确输出难以获取和确定的问题. 蜕变测试是通过构造被测对象的输出与测试输入之间相应的变化关系来解决预言缺失的问题, 这种关系被称为蜕变关系, 通常由研究者自行定义[39]. 差分测试是通过向一系列基于相同规约实现的应用程序提供相同的输入来观察他们的输出是否一致, 若不一致则认为输出存在错误[40]. 在测试用例的生成过程中, 受传统软件测试中覆盖率指标的启发, 研究者们提出了多种面向智能软件系统的测试覆盖度量指标, 用于指导测试过程. 其他常见的提升测试效率的方法还包括基于统计学设计的测试优先级排序方法和基于遗传算法的测试用例检索方法.

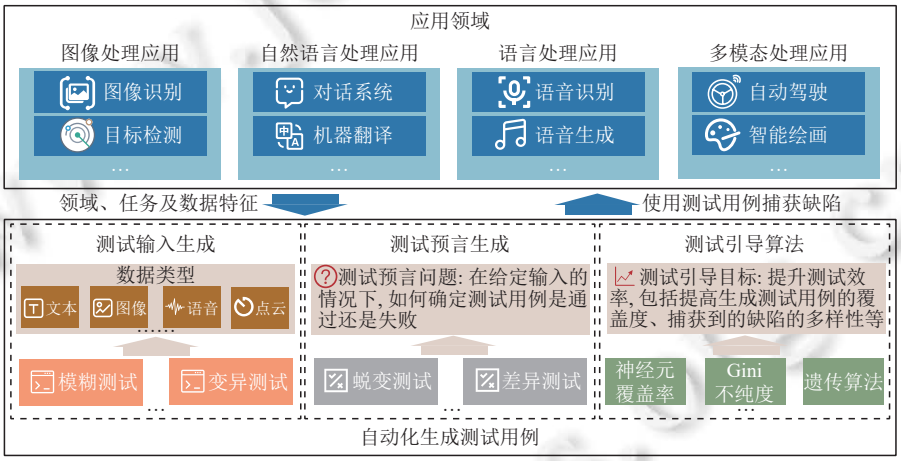

图 1 面向智能软件系统的测试用例生成方法研究框架

由于不同应用领域的智能软件系统在输入数据特征和任务目标上存在显著差异, 这些差异直接影响测试用例生成方法的设计与适用性. 因此, 从应用领域视角对测试用例生成方法进行分类和分析, 能够揭示领域特定的测试需求和挑战, 并为研究人员设计针对性测试方法提供指导. 为此, 本文在归纳相关文献的基础上, 基于智能软件系统的应用领域及任务目标, 对测试用例生成方法进行系统梳理与总结, 以填补现有综述在领域针对性分析及应用适配性分析方面的空白, 并构建更具实践意义的研究框架. 首先, 本文将智能软件系统按照人工智能技术的应用领域进行初步划分, 并根据系统实现的任务目标进行进一步归纳. 这样的分类方式不仅揭示了不同测试方法在应对特定领域挑战中的适配性, 还通过任务目标的进一步归纳, 体现了测试需求与系统功能的紧密关联.

具体来说, 智能软件系统可分为图像处理应用、自然语言处理应用、语音处理应用、点云处理应用以及多模态数据处理应用等. 这些领域划分涵盖了当前人工智能技术在智能软件系统中的主要应用方向, 并有效反映了不

同领域的测试需求与挑战. 在分类中, 前 4 种类型的智能软件系统主要处理单一模态的数据, 而多模态数据处理应用能够同时接收和处理多种类型的数据. 例如, 自动驾驶系统需要通过汽车上的摄像头、激光雷达等多种传感器收集环境和车辆自身的多模态信息, 以在复杂多变的应用场景中实现高准确率和高安全性的目标[41]. 此外, 部分行业特定领域 (如金融风控、医疗诊断) 虽未单独列为分类, 但其测试需求通常可以归为多模态或特定单模态处理的变种. 因此, 在测试方法层面, 这些行业领域的需求已被纳入上述分类的框架之中, 从而保证了分类的完整性和覆盖广度.

2 文献收集与汇总

本文采用如下步骤对文献进行了收集、筛选与分类, 具体流程如图 2 所示.

首先, 本文定义文献纳入标准为: 文献需涉及面向某类智能软件系统的测试用例生成方法, 并公开发表于会议、期刊、技术报告、书籍或收录于存储库的预印本中. 针对 2012 年 1 月 1 日–2024 年 12 月 31 日期间公开的相关文献, 本文使用搜索关键词在国内外知名数据库与学术引擎中进行检索, 随后逐一检查收集文献的参考文献, 以发掘与搜索关键词相关的补充文献. 具体的文件搜索步骤如下所示.

(1) 使用 “object detection testing”“machine translation testing”“machine reading testing”“dialogue system testing” “speech recognition testing”“autonomous vehicle testing”“self-driving cars testing”“deep learning testing”等关键词进行文献检索, 并在线搜索 ACM Digital Library、IEEE Xplore Digital Library、Spring Link Online Library、Wiley Online Library、Elsevier ScienceDirect、ProQuest Research Library、arXiv、中国知网和万方数据库等国内外的知名数据库和学术搜索引擎, 并在标题、摘要、关键词和索引中进行检索 (G1).

(2) 对于上述步骤所获得的文献, 逐一审查其参考文献, 识别与搜索相关的文献并纳入文献集合, 即从已经获得的文献集合的参考文献当中识别遗漏的文献.

综上两个步骤, 本文收集到与搜索关键词相关的文献共 240 篇 (G2). 随后, 本文对这些文献进行了严格的人工过滤与筛选. 通过分组审阅的方式, 手动过滤了与面向智能软件系统测试用例生成方法不相关的文献. 审阅过程中采用交叉分组审查, 对于每一篇拟删除的文献, 需经研究小组一致审议后方可删除. 删除文献的具体步骤如下.

(1) 在文献集合 G2 中, 删除与软件测试领域无关的 74 篇文献, 得到包含 167 篇文献的集合 G3.

(2) 在文献集合 G3 中, 删除与深度学习框架测试相关的 32 篇文献, 得到包含 135 篇文献的集合 G4.

(3) 在文献集合 G4 中, 删除 11 篇综述与实证研究类文献, 得到包含 124 篇文献的集合 G5.

(4) 在文献集合 G5 中, 删除与对抗样本生成相关、与测试用例生成无关的 44 篇文献, 最终得到包含 80 篇文献的集合 G6, 作为本文最终梳理总结的文献列表.

最终, 本文共筛选出 80 篇与研究主题相关的文献. 在筛选过程中, 本文剔除了与对抗样本相关的文献, 主要原因在于对抗样本生成与测试用例生成的目标和方法存在本质区别. 对抗样本生成的核心目标是通过设计微小但有意的扰动, 干扰模型决策或诱导其产生错误输出, 更多关注的是模型在极端或边界条件下的鲁棒性与安全性, 而非系统在实际应用场景中的表现[42]. 相比之下, 测试用例生成的目标更广泛, 主要是验证系统在多样化输入条件下的功能性、可靠性和表现一致性. 测试用例生成通常强调输入数据的多样性和覆盖性, 旨在通过模拟现实应用中的输入场景 (包括正常和异常情况) 评估系统在实际任务中的行为是否符合预期, 而非依赖特定的攻击性算法或梯度信息, 这决定了其与对抗样本生成方法的根本区别. 基于上述差异, 本文认为对抗样本生成更侧重于评估模型的安全性与攻击防护能力, 而非软件测试领域中的功能性和可靠性验证[43]. 由于本文综述的核心目标是总结测试用例生成方法及其在智能软件系统中的应用, 决定删除与对抗样本生成相关的文献, 以避免主题混淆, 保持综述内容的聚焦性与系统性.

随后, 由熟悉智能软件测试用例生成的研究人员同样采用交叉分组的方式, 对 80 篇文献进行具体的分类, 文献分类的总览图如图 3 所示. 本文根据智能软件系统的应用领域, 将所收集文献划分为 6 大类: 面向图像处理应用、自然语言处理应用、语音处理应用、点云处理应用、多模态数据处理应用, 以及面向深度学习模型的测试用例生

成方法相关文献. 其中, 图像处理应用细分为图像识别与目标检测, 自然语言处理应用细分为对话系统、机器翻译与机器阅读理解. 此外, 本文还总结了面向深度学习模型的测试用例度量指标与选择方法.

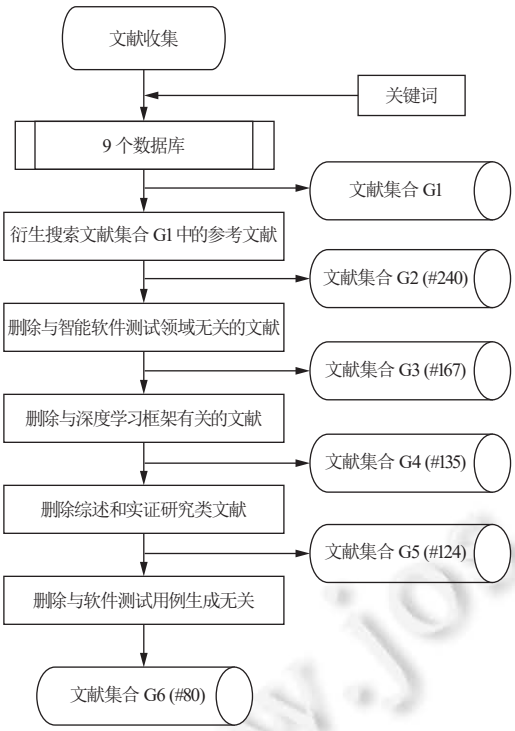

图 2 文献收集和汇总的流程图

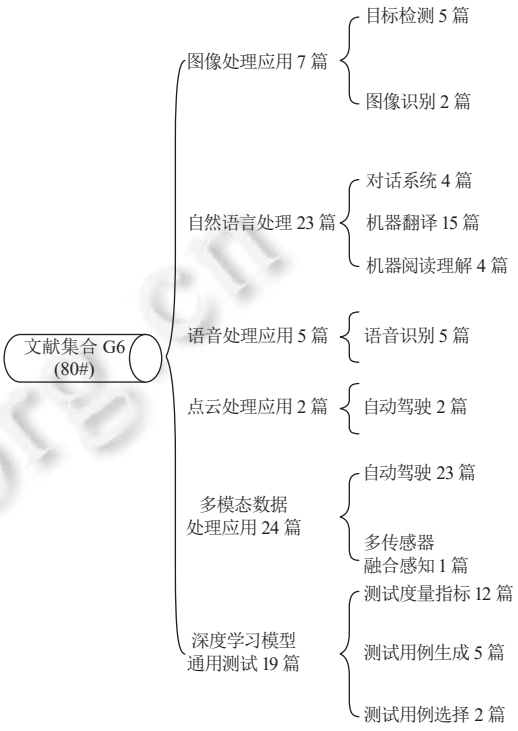

图 3 文献分类总览图

在本文所总结的 80 篇文献中, 65 篇属于会议论文, 10 篇属于期刊论文, 5 篇收录于 arXiv 预印本, 涵盖了软件工程、人工智能等领域的国际顶级会议与期刊, 具有较高的权威性. 具体而言, 这些文献涵盖的会议和期刊分别来自 7 个领域: 软件工程 (62 篇)、人工智能 (5 篇)、分布式系统 (1 篇)、系统软件 (1 篇)、计算机图形学与多媒体 (2 篇)、智能交通 (2 篇)、信息物理融合系统 (2 篇). 本文所收集到的文献分布领域如图 4 所示, 所有文献的发表时间分布表如图 5 所示.

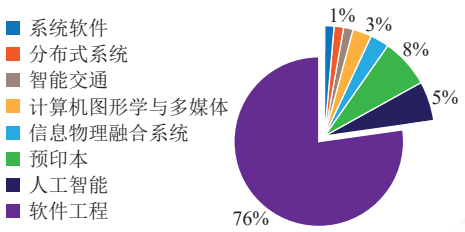

图 4 文献领域分布图

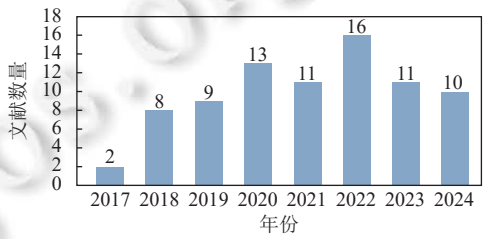

图 5 文献发表时间分布表

3 图像处理应用

图像处理是将图像转换为数字形式后, 进行特定操作以生成新的图像或提取有效信息的过程[44]. 随着环境、军事、医疗和工业等领域应用需求的增长, 图像处理技术在许多应用领域受到了广泛的关注并取得了突破性的进展[45−47]. 通常来说, 图像处理应用首先需要将空间分布和亮度取值均连续分布的模拟图像经采样和量化转换成数字图像, 再对数字图像进行预处理, 以增强有关信息的可检测性并最大程度地简化数据, 从而提升后续处理的可靠

性. 之后, 应用软件会使用卷积神经网络等人工智能模型对预处理的图片进行特征提取、预测分类等操作, 最终输出一张新的图像或包含有效信息的分析结果[48]. 其中所集成的人工智能模型的类别及功能与该应用需要完成的图像处理任务密切相关.

在图像处理领域, 测试用例生成主要面临以下挑战: (1) 目标动态变化的模拟: 为了验证被测对象对不同空间布局的理解能力以及对视角和形状变化的鲁棒性, 测试用例需模拟出包含多种空间布局的场景动态变化过程, 同时确保变异输入的语义不失真; (2) 复杂场景图像的多样化仿真: 为了评估被测对象在噪声或背景干扰条件下的准确性, 测试用例需仿真不同复杂程度与质量等级的图像场景, 并确保生成场景在语义上相关, 且变异幅度合理可控; (3) 基于图像语义特征的测试预言构建: 由于图像处理任务的结果与图像语义特性密切相关, 构建测试预言需精确定位输入变化与预期输出之间的映射关系, 特别是在动态场景和复杂变异条件下的测试验证中.

当前, 测试领域的研究人员主要关注的两个图像处理任务是图像识别和目标检测. 这两个任务是计算机视觉领域的核心任务, 也是许多图像处理应用的基础. 为解决上述挑战, 研究者们基于这两个图像处理任务的特性提出了多种测试用例生成方法 (如表 1 所示). 图 6 展示了面向图像处理应用的测试用例生成方法研究框架. 针对现有测试挑战, 研究者们设计了基于神经网络内部行为覆盖分析与基于图像特征或领域知识变异的技术路径, 以生成涵盖多种场景和语义特征的输入图像数据, 此外, 通过测试预言的构建, 精准验证输入变异对神经元行为或输出语义的影响, 从而评估图像处理系统的可靠性和稳健性. 接下来, 本文将按照图像处理任务对现有方法进行详细分析.

表 1 现有面向图像处理应用的测试用例生成方法总结

应用任务	发表年份	文献来源	测试输入生成	测试预言构建
图像识别	2020	[49]	(1) 分割变异; (2) 融合变异	蜕变关系: 图像变异前后的识别结果不发生变化
	2020	[50]	获取测试输入对应的神经元激活向量	混淆错误: 混淆分数小于设定阈值 偏差错误: 偏差分数大于设定阈值
目标检测	2017	[51]	排列基本对象, 调整图像参数, 保留物体纵横比	工具可获取到生成图像中包含的对象, 并根据设定的指标提供测试结果
	2018	[52]	(1) 改变亮度和对比度; (2) 平移缩放、水平剪切、旋转; (3) 模糊; (4) 模拟雾效果和雨效果	蜕变关系: 图像变异前后的检测结果不发生变化
	2020	[53]	将给定的真实图像作为背景, 再将对象实例插入背景	蜕变关系: 在不考虑插入对象的检测结果时, 原始图像和合成图像的目标检测结果是等价的
	2021	[54]	将背景图像进行三维建模, 采用启发式方法插入新对象	蜕变关系: 在原始图像中被检测到的目标应该在新合成的图像中仍然被检测到
	2022	[55]	从图像中移除对象, 再使用随机像素噪声填充被移除的位置	蜕变关系: 随机移除原始图像中检测到的对象之一并不应该改变剩余对象的检测结果

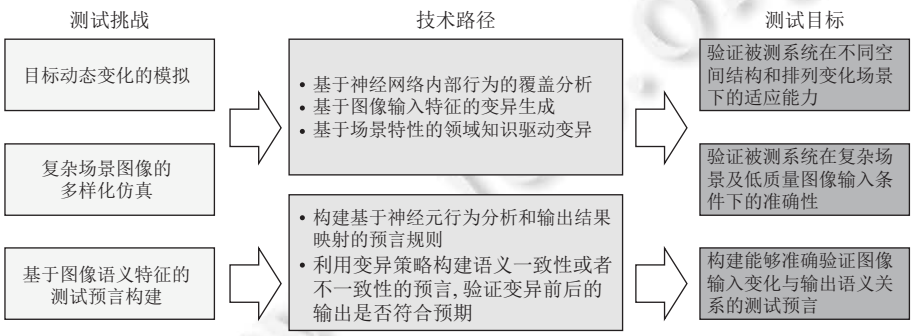

图 6 面向图像处理应用的测试用例生成方法研究框架

3.1 图像识别

图像识别是指利用计算机自动识别图像中物体、场景、人物、文字及动作等内容, 并对图像进行处理、分析与理解的过程. 该任务通常分为两个步骤, 图像特征提取和图像分类预测. 随着计算机技术及硬件的发展, 图片分

辨率持续提高, 传统图像识别方法已难以有效处理大规模高分辨率图像. 为解决该问题, 深度学习技术已经被用于完成与图像识别相关的各项任务, 比如人脸识别、车牌识别和医学影像识别等. 在被用于图像识别的深度神经网络中, 卷积神经网络的应用最为广泛. 该神经网络基于图像任务的平移不变性设计, 通过卷积层与池化层高效提取图像特征, 同时显著降低参数量并提升计算效率.

针对当前图像识别技术的特征, 现有研究者们提出了基于卷积神经网络内部结构的白盒测试方法, 该类方法能够有效洞悉在输入不同图像时被测对象核心模型内部显性的数据流动及变化过程. 根据这些显性清晰的信息, 研究者们能够通过特定的处理方法获取到在输入具有不同特征的图像时被测对象的即时反馈, 以进一步地达成预设的测试目标. 该类方法在测试用例生成中, 侧重于构建能够识别特定类型错误的测试预言, 以实现与数据特征密切相关的测试目标. 例如, 在测试方法 DeepInspect 中, 测试用例生成方法的重点在于如何构建测试预言, 以分辨出图像识别输出中的类混淆和类偏差错误[50]. 类混淆错误是指被测对象将一个类别与另一个类别混淆, 类偏差错误是指被测对象在某些情况下的输出偏向于某个类别. 对于每个图像类别, DeepInspect 都会创建一个神经元概率向量, 再计算不同向量之间的欧几里得距离去与混淆阈值和偏差阈值进行比较, 以捕获和区分这两种类型的错误.

此外, 还有一些研究者提出了关注输入图像特征多样性的黑盒测试方法. 与白盒测试不同, 该类方法不考虑被测对象的内部结构与特性, 而重点验证其是否能按预期输出正确识别结果. 为了提升输入图像的多样性, 当前方法会选择合适的策略去对种子图像进行变换. 常见的图像变换策略包括基于单张图像的几何变换、颜色空间变换和像素点操作等, 也包括基于多张图像的图像拼接等[56]. 由于需保证生成图像的真实性, 基于多张图像的变换策略相较于单张图像的变换, 实施难度更大. 为应对这一挑战, Wang 等人[49]提出了一种基于领域知识的图像变异方法. 该方法首先从给定图像中提取并分类关键特征, 然后针对不同特征实施分割变异或与嵌入图像融合的伪装性变异. 分割变异通过领域知识指导, 自动分割出可变异的图像成分; 融合变异则基于亮度量化结果, 确定原始图像与嵌入图像的连通区域, 并调整亮度分布, 以生成高质量的伪装图像.

3.2 目标检测

目标检测是指在数字图像或视频中识别并定位特定类别语义对象实例 (如人、建筑物和汽车等)[57], 不仅需识别对象类别, 还需准确定位其在图像中的位置[58]. 随着深度学习的快速发展, 基于深度学习的目标检测技术已逐步取代传统的依赖人工特征提取与分类的检测方法[59]. 许多计算机视觉任务, 如人脸识别、目标追踪和图像分割, 均基于深度学习的目标检测技术实现. 因此, 目标检测技术的可靠性对于计算机视觉的发展及应用至关重要. 研究表明, 微小且视觉上难以察觉的扰动可导致目标检测模型产生错误的分类结果或定位偏差[60]. 目前, 目标检测技术已经被运用于自动驾驶系统中的感知模块, 许多研究人员结合目标检测的任务特性和自动驾驶的领域特性提出了多种自动化测试用例生成方法.

当前, 针对图像目标检测任务的测试用例生成方法, 主要可分为两类: 基于完整图像的变异与基于目标对象的局部修改. 第 1 类方法通常结合自动驾驶应用场景特点, 设计图像变异策略, 以模拟汽车行驶过程中可能捕捉到的环境图像. 随后, 通过对比图像变换前后的识别结果, 检测可能引发致命事故的错误行为. 与用于图像识别测试的变异策略相比, 用于自动驾驶视觉感知模块测试的变异策略会着重考虑天气、汽车运动等给环境图像带来的影响. 以自动驾驶系统测试工具 DeepTest 为例, 通过在种子图像上应用改变亮度或对比度、平移缩放、水平剪切、旋转、模糊、模拟雾效果和雨效果等转换算子, 以模拟摄像机镜头畸变、物体运动及多种天气条件[52]. 为提升测试效率, DeepTest 以神经元覆盖率作为输入空间划分指标, 将空间划分为不同等价类, 并从每个等价类中选取样本以尽可能覆盖整体空间. 在图像转换过程中, DeepTest 使用了贪婪搜索技术, 以高效寻找提升神经元覆盖率的图像变换组合.

第 2 类方法可根据对图像中对象的具体操作类型进一步细化, 具体操作类型包括插入对象和移除对象, 不同的对象操作类型对应不同的蜕变关系. 例如, 插入新对象后, 检测结果中应包含对应标签, 且不应影响图像中原有对象的检测结果. 类似地, 移除对象后, 其余对象的检测结果也应不被影响. 为了保证生成图像的真实性, 插入对象类的测试用例生成方法需要着重考虑对象的插入位置、大小和合成图像参数等配置. 该类方法的基本流程包括: 1) 确定背景图像与插入对象; 2) 确定插入位置; 3) 完成图像合成; 4) 依据蜕变关系捕获错误检测结果. 具体来说, Dreossi 等人[51]提出了用于测试自动驾驶感知模块的图像生成框架, 该框架的目标是生成真实的图像, 而不是在已

有的图像中引入扰动. 该框架由 3 个主要模块组成: 图像生成器、采样方法和可视化工具. 图像生成器通过排列基本对象 (如道路背景、车辆), 调整亮度、对比度与饱和度等图像参数, 并保留物体纵横比, 从而渲染生成真实道路场景图像. 基于主动优化的采样方法能够均匀地覆盖修改空间, 目标是为图像生成器提供修改点以生成能够被错误分类的图片, 可视化工具根据设定的指标提供测试结果. 与之类似, MetaOD[53]首先收集了许多按类别区分的对象集, 并基于设定标准, 从与背景图像相关的对象集中筛选对象实例. 图像合成阶段, MetaOD 使用受 delta 调试启发的领域特定标准和技术来找到可能触发预测错误的位置, 并将对象插入.

此外, 还有的研究者着重评估自动驾驶系统中的感知模块对于其他交通参与者的检测性能, 重点关注被插入参与者对被测对象的影响. 例如, Shao[54]提出的一种通过对真实图像中车辆进行三维重建来生成测试用例的方法. 该方法首先确定一个被插入的对象以及一个背景图像, 随后对背景图像进行三维建模, 并采用启发式方法完成对象插入与图像合成. 该启发式方法使得边框和目标分割所得的物体之间不产生任何重叠. 与其他工作不同的是, 该方法会根据输入的图像重现插入车辆的三维模型. 三维重建由边界估计和消失点估计两部分组成, 前者是为了识别每个物体的边界, 后者是为了识别所有车辆和其他物体的视角.

与插入对象不同的是, 移除对象的测试生成策略需要考虑如何填充被移除对象对应的空缺位置, 同时需要保证生成图像的语义正确, 即对剩余对象的语义不产生影响. 例如, Wang 等人[55]所提出的方法通过从给定的原始图像中移除对象来构造测试输入, 再通过比较对象移除前后的目标检测结果来判断被测对象是否出现检测错误. 该方法从图像中获取到可被移除的对象的位置和类别信息等, 并使用随机像素噪声填充被移除对象的位置. 随机像素噪声并不会对图片中的其他语义成分产生影响, 极大地降低了该方法的误报率.

3.3 现有方法优缺点分析

随着近年来自动驾驶技术的快速发展, 面向图像处理应用的测试用例生成方法亦受到广泛关注. 然而, 经总结分析, 本文发现现有方法存在以下问题与局限性: (1) 应用场景的局限性: 现有方法主要聚焦于自动驾驶系统的视觉感知模块, 通过生成或变异输入图像模拟驾驶场景中的复杂情况. 然而, 当这些方法应用于其他类型的被测对象时, 其有效性还有待考证. 这一局限性主要源于现有方法在测试用例生成过程中对领域特定知识的高度依赖, 缺乏跨领域适配的通用性与灵活性. (2) 测试输入真实性的不足: 部分方法通过调整整体图像参数以尽量降低变异对关键对象的影响; 另一些方法则采用拼接或组合真实图像的方式生成接近真实世界的新图像. 然而, 多数方法未对生成图像的有效性进行人工审核, 测试用例的真实性及不同图像变换方法对误报率的影响仍有待进一步验证与研究. 如表 2 所示, 本文结合各技术路径进行了更为详细的优缺点分析.

表 2 现有面向图像处理应用的技术路径优缺点总结

技术路径	优点	缺点
基于神经网络内部行为的覆盖分析	(1) 利用神经元激活状态引导输入生成过程, 提升模型内部行为覆盖率, 以便更全面发现模型在不同输入特征下的潜在缺陷 (2) 不依赖具体语义标签, 即从被测对象的行为特征出发进行输入空间划分, 提升测试用例生成的引导性和覆盖性	(1) 依赖于被测对象内部模型的结构信息, 限制在白盒测试场景下使用 (2) 神经元行为通常缺乏可解释性, 限制了测试结果的直观性和可用性
基于图像输入特征的变异生成	(1) 变异策略丰富, 易于实施, 可生成多样化的测试输入去模拟现实应用环境 (2) 无需了解被测对象的模型内部结构, 适用于多种黑盒系统	(1) 难以系统性覆盖高阶语义或边界性场景 (2) 图像真实性要求限制了变异操作自由度
基于场景特性的领域知识驱动变异	(1) 具备更强的上下文建模能力, 可用于复杂的测试场景当中 (2) 有效提升测试用例的合理性与有效性, 减少误报率	(1) 测试生成质量高度依赖场景建模的准确性, 领域知识存在偏差易引发误判 (2) 变异能力受限于知识库的覆盖范围, 容易导致测试盲区, 难以触发缺陷
基于神经元行为分析的测试预言构建	能基于神经元行为与输出间的关系, 构建有针对性的测试预言, 有助于发现特定类别的错误	(1) 依赖神经元输出模式, 泛化能力有限 (2) 神经元行为与输出结果之间的因果关系难以验证
基于图像语义的测试预言构建	(1) 更易于人类理解, 适合高层语义的验证 (2) 不依赖被测对象的内部结构信息, 可用于黑盒测试	(1) 语义差异判断标准难以统一, 常需依赖人为设定的阈值 (2) 语义提取通常依赖外部模型, 可能引入误差干扰

3.4 小 结

综上所述, 当前面向图像处理应用的测试用例生成方法在测试效果提升方面已取得一定进展, 但仍存在真实性保障不足、变异策略适配性有限等问题. 针对上述问题, 本文认为今后研究人员在设计面向图像处理应用的测试输入方法时, 需更加注重结合图像数据特性, 重点保障测试输入的真实性, 尤其是确保被识别或被检测对象无失真变形, 以避免蜕变关系失效及测试误报. 此外, 未来研究可重点关注测试用例变异过程中不同变换对测试结果的影响机制, 分析易引发误报的变异类型, 并设计针对性的控制策略. 同时, 如何通过总结现有技术路径的不足并进一步改进方法, 以拓宽适配的应用场景, 也是未来的重要研究方向.

4 自然语言处理应用

自然语言处理 (NLP) 是指使计算机能够理解、分析和处理人类自然语言的技术, 其核心目标是实现人类语言与机器语言之间的相互转换. 自然语言处理可以细分为两个子领域: 一是围绕基本问题展开的核心研究领域, 如语言建模、词法分析、句法分析与语义理解等; 二是面向实际应用场景的问题领域, 如命名实体识别、机器翻译、阅读理解等[61]. 通常, 自然语言处理应用需要首先成功解决一个或多个核心领域的基础任务, 再在此基础上进一步实现具体的应用功能. 这些应用程序通常先利用自然语言理解技术, 将含有杂质、无序且不规则的自然语言文本, 转换为规则化、标准化、便于后续处理的结构化文本; 随后, 通过词袋模型 (bag-of-words)、词向量 (word embedding) 等文本表示方法, 将文本转化为计算机能够理解和运算的形式, 进而完成分类、推理或生成等后续处理. 这类应用的输入数据普遍具有语义复杂性、上下文敏感性及结构多样性的特征, 给测试用例生成带来了显著挑战.

在自然语言处理领域, 测试用例生成主要面临以下几个挑战: (1) 多样化输入特征的文本生成: 测试用例需要能够覆盖多种输入特征, 例如文本长度、语言风格、语法结构与句型复杂度, 以满足不同自然语言处理任务的测试需求, 并提升测试输入的多样性与覆盖范围; (2) 文本语义和质量的可控: 在测试用例生成过程中, 需确保文本在语义变化程度、语法正确性、语言流畅性及风格连贯性等方面具有可控性和合理性, 以保证生成文本既符合实际应用情境, 又具备有效触发被测系统特定行为的能力; (3) 基于上下文语义的测试预言构建: 测试预言的构建需要结合具体任务特性与上下文预期语义变化, 设计可自动验证的规则、判别标准或度量机制, 从而减少对人工审核的依赖, 提升测试效率, 并降低由于主观性判断导致的偏差或误差.

如表 3 所示, 随着自然语言处理应用在智能对话系统、机器翻译系统及机器阅读理解系统等领域的广泛部署与应用, 研究人员逐步提出了针对这些具体任务的测试用例生成方法. 通过对现有文献的系统调研, 本文发现, 当前研究主要集中在智能对话 (intelligent dialogue system)、机器翻译 (machine translation) 和机器阅读理解 (machine reading comprehension) 这 3 个典型任务. 这些任务覆盖了自然语言理解与生成的多个关键环节, 测试用例生成方法也在应对不同输入输出形式、多轮交互语境以及复杂推理能力的测试需求方面, 体现出不同的设计思路与技术挑战.

表 3 现有面向自然语言处理应用的测试用例生成方法总结

应用任务	发表年份	文献来源	测试输入生成	测试预言构建
智能对话	2021	[62]	(1) 同义词替换; (2) 回译; (3) 插入单词	蜕变关系: 自然语言理解模块对于语义相同的测试用例的输出应该是相同的
	2023	[63]		
	2024	[64]		
	2024	[65]	(1) 对话轮次打乱; (2) 对话轮次减少; (3) 对话轮次复制; (4) 对话轮次打乱并减少; (5) 对话轮次打乱并复制	蜕变关系: 除不可回答问题外其他问题的答案在扰动前后应该与原始问题的答案相似; 使用不同扰动算子生成可回答性相同的问题的答案应相似; 基于同一原始问题生成的可回答性不同的问题的答案应该不同
机器翻译	2020	[66]	使用词性相同的词语替换句子中的词语	蜕变关系: 语义相似的语句的翻译结果应该具有相似的语法结构
	2021	[67]	从源句中提取名词短语	蜕变关系: 名词短语在不同的上下文中应该具有相似的翻译

表 3 现有面向自然语言处理应用的测试用例生成方法总结 (续)

应用任务	发表年份	文献来源	测试输入生成	测试预言构建
机器翻译	2020	[68]	(1) 使用词性相同但词义不同的词语替换原词语 (2) 删去原句中有意义的一个单词或者一个词组	蜕变关系: 具有不同语义的句子不应该具有相同的翻译结果
	2018	[69]	选定中间语言, 获取到直接翻译和间接翻译	蜕变关系: 直接翻译和间接翻译的翻译结果一致
	2019	[70]	通过爬虫构造双语语料库	基于语料库检查正确翻译结果是否存在以及统计出现的次数
	2020	[71]	基于上下文相似度替换源句中的词语	采用 TF-IDF、BLEU、LCS、 ED等方法来计算翻译之间的相似度, 相似度小于阈值则存在错误
	2022	[72]		
	2020	[73]	(1) 注入大写或小写类型的错别字; (2) 字母全部转成大写; (3) 省略句号; (4) 修改标点符号; (5) 句子尾部注入微小噪音; (6) 改变句子的主语; (7) 改变句子的宾语; (8) 用语义相似的动词替换源句的动词	蜕变关系: 源句转换前后的翻译结果不应该发生显著变化
	2021	[74]	通过句子压缩模型获取到包含句子基础元素的模板和修饰语, 再将变换后的修饰语按序插入模板	蜕变关系: 添加修饰语后句子的语义结构主干不应该发生变化, 并且添加前后存在结构路径的包含关系
	2022	[75]	将给定的文本或句子翻译成中间语言, 然后将结果翻译回源语言	计算基于正则表达式的相似性、基于确定性有限自动机的相似性和混合相似性, 如果相似度低于预期阈值, 则认为存在错误
	2023	[76]	(1) 性别调换; (2) 肯定/否定调换; (3) 单复调换; (4) 时态调换	基于多义词的词义库判断变异前后同一个多义词的译文在词义库中的一致性
	2024	[77]	(1) 含义插入; (2) 部分词语替换	基于结果一致性的蜕变关系: 在原文上下文中添加术语的适当参考, 应不影响术语和原文的翻译结果 基于结果不一致的蜕变关系: 修改多词术语的一部分, 应导致翻译结果的相应变化
	2024	[78]	修改源句中的人口群体属性, 具体包括性别和国籍	蜕变关系: 修改人口群体属性, 不应该改变与这些属性无关内容的翻译结果
	2024	[79]	基于依存关系裁剪源句	蜕变关系: 未裁剪到核心语义的句子的翻译结果应该与源句的翻译结果保持较高的相似程度
	2024	[80]	采用现有的基于结构和基于文本的机器翻译测试输入生成方法, 并生成源输入、变异输入及对应翻译之间在词闭包层面的关联关系	蜕变关系: 含义相似 (不同) 的输入变化词应被翻译为语义相似 (不同) 的内容
阅读理解	2021	[15]	(1) 使用反义形容词; (2) 时态改变; (3) 主动和被动语态互换; (4) 添加否定词; (5) 使用同义形容词; (6) 改变状语从句位置; (7) 语态变化	基于反义输出的蜕变关系: 输出应与原始输出反义 基于同义输出的蜕变关系: 输出应与原始输出同义
	2021	[81]	(1) 由现有wh-疑问句的答案衍生出新的wh-问题 (2) 由现有wh-问题的答案衍生出新的一般疑问句 (3) 由现有一般疑问句或选择疑问句的答案衍生出新的wh-问题	蜕变关系: 如果源输出和后续输出都是正确的, 那么源输出与源问题构成的“伪事实”应该等价于后续输出与后续问题构成的“伪事实”
	2022	[82]	(1) 插入冗余子句; (2) 合并两个问句; (3) 将两段上下文合并为一段文本	蜕变关系: 当变换前后问句和上下文都等价时, 输出答案的语义应该一致
	2022	[83]	(1) 整句反转翻译; (2) 状语前置; (3) 单词插入; (4) 同义词替换; (5) 实体别名替换; (6) 单词缩写替换; (7) 键盘临近字母输入错误; (8) 单词错误输入; (9) OCR扫描识别错误; (10) 重复问号	蜕变关系: 当问题的语义不发生改变时, 输出答案的语义同样不应该发生改变

根据表 2 的内容, 并结合现有自然语言处理应用中测试用例生成所面临的挑战, 本文总结出如图 7 所示的面向自然语言处理应用的测试用例生成方法研究框架. 为应对当前存在的挑战, 现有方法主要围绕文本变异生成、输入质量控制与测试预言构建这 3 个方面展开. 文本变异方法基于句法结构、语义特征及任务特性, 旨在生成多样化的输入文本, 以覆盖不同输入特征; 输入质量控制则通过语法正确性检测、语义变异幅度控制与文本适用性筛选等策略, 确保测试数据的合理性与公平性; 测试预言构建方面, 研究者们依托人工或自动化语料库、句法变异

影响分析及语义变化趋势建模, 设计了可自动验证输入变异对系统输出影响的规则或度量标准. 接下来, 本文将按照具体自然语言处理任务, 进一步细致地梳理和分析现有测试用例生成方法.

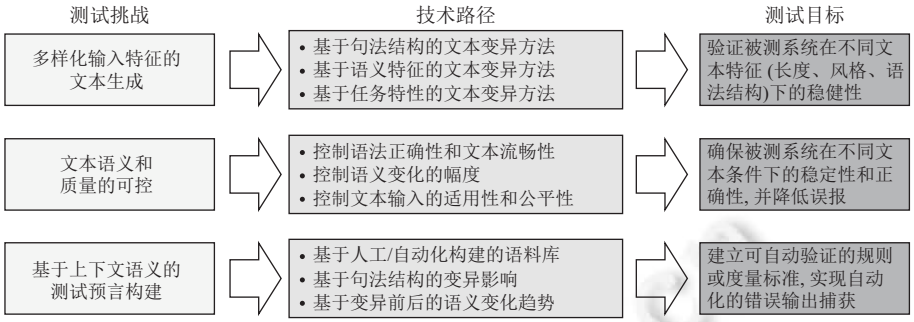

图 7 面向自然语言处理应用的测试用例生成方法研究框架

4.1 智能对话

智能对话系统通过运用自然语言理解、自然语言生成和对话管理等技术, 实现与人类通过自然语言进行交互, 主要分为面向任务的对话系统和开放域对话系统[84]. 面向任务的对话系统专为特定领域或任务设计, 如航班预订、酒店预订、客户服务和技术支持等; 而开放域对话系统则旨在支持关于任意主题的对话, 其最终目标是使人机交互更加接近人与人之间的自然交流. 智能对话系统需要准确识别用户意图, 关注交互方式与情感感知, 从而实现对用户问题的精准简洁回答. 本文仅讨论输入输出均为文本的智能对话系统, 不考虑输入输出为语音的系统类型.

在基于文本的对话系统中, 文本处理流程通常由自然语言理解模块、对话管理模型和自然语言生成模块共同完成. 目前, 已有研究者针对上述流程中的自然语言理解模块设计了多种测试用例生成方法. 自然语言理解模块主要承担槽位填充 (slot filling) 和意图检测 (intent detection) 两项任务. 意图检测将输入句子分类至相应的意图类别, 有助于缩小后续模块的处理范围; 槽位填充则是在输入文本中识别关键单词并填充至预定义的语义槽位中. 针对这两个任务的特性, DialTest 基于模糊测试理论使用了 3 种文本变异算子对种子文本进行变异以生成新的测试输入, 并基于自然语言理解模块中语义相同输入应具有一致输出的核心思想, 设计了相应的蜕变关系[62]. 3 种文本变异算子分别为: 使用同义词替换种子文本中的词语; 通过机器翻译将种子文本翻译至中间语言后再译回原语言; 使用语言模型在种子文本中插入单词. 为了提高测试效率, DialTest 采用了 Gini 不纯度[85]来引导测试用例生成过程. 然而, Shen 等人[63]和 Chen 等人[64]指出, DialTest 在测试用例生成过程中将所有的种子文本平等看待, 导致在无法检测到缺陷的种子上消耗大量的资源. 针对这一问题, 他们分别提出了改进方法——EcoDialTest 和 DialTest-EA. EcoDialTest 优化了基于 Gini 不纯度的测试用例生成过程, 它按照种子的初始 Gini 不纯度对种子进行分类, 不同类别的种子文本会采用不同的突变策略以提高新生成测试用例的 Gini 不纯度. DialTest-EA 同样是优化了基于 Gini 不纯度的测试用例生成过程, 采用了基于蚁群优化算法 (ant colony optimization, ACO) 的轻量级变异能量调整策略, 能够自适应地调整各种子的变异优先级, 使潜在有效的种子有更多机会生成后续测试输入. 实验结果表明, EcoDialTest 和 DialTest-EA 均能生成比 DialTest 更多能够发现缺陷的测试用例, 但两者在性能上的优劣尚有待进一步比较.

随着大语言模型的广泛应用, 研究者们开始探索如何面向基于大语言模型的对话系统设计有效的测试用例生成方法. 此类系统具有高度依赖上下文、多轮交互频繁、易受训练数据污染影响, 以及输出内容不确定性高等特点. 上述特点要求在生成测试用例时, 充分考虑上下文连贯性与一致性, 并设计能够验证多轮对话逻辑及语义理解能力的测试预言. 为此, Guo 等人[65]提出了基于多轮对话的蜕变测试工具 MORTAR. 给定原始种子测试用例, MORTAR 首先通过 5 种对话级扰动 (对话轮次打乱、对话轮次减少、对话轮次复制、对话轮次打乱并减少、对话轮次打乱并复制) 生成问题序列, 并对每轮问题进行可回答性检查并分配预期答案. MORTAR 包含了 3 种可回

答性检查方式: (1) 基于语义的检查: 分析对话中代词、隐含信息等与其他实体 (具有明确指代意义的对象) 之间的关系, 判断能否从给定的对话上下文中找到问题的答案. (2) 基于本体的检查: 利用基于知识图谱的对话信息模型, 将对话中实体和关系构建知识图谱, 通过对比原始问题、可独立回答问题及上下文信息图中实体和关系, 判断问题能否独立回答或依赖上下文回答. (3) 面向特定任务的检查: 针对阅读理解任务, 即使某个代词在上下文中未明确被提及, 只要其在上下文中出现过, 就会被视为可回答. 如果生成的问题通过了可回答性检查, 那么预期答案就是原始标注答案; 如果问题不可回答, 预期答案则为 “Unknown”. MORTAR 方法构建的 3 种蜕变关系为: (1) 除不可回答问题外, 其他问题的答案在扰动前后应该与原始问题的答案相似; (2) 使用不同扰动算子生成可回答性相同的问题的答案应相似; (3) 基于同一原始问题生成的可回答性不同的问题的答案应该不同. 实验结果表明, 基于上述蜕变关系, MORTAR 能够高效且全面地检测基于大语言模型对话系统中的各类潜在缺陷.

4.2 机器翻译

通常, 输入到机器翻译系统的语言被定义为源语言, 输出的语言被定义为目标语言. 因此, 机器翻译可描述为是将源语言词语序列转换为目标语言词语序列的任务[23]. 机器翻译可按实现方法分为基于规则、基于统计和基于神经网络的 3 类方法, 其中基于神经网络的方法具有当前最先进的性能. 尽管基于神经网络的机器翻译技术已达到较高水平, 但其输出仍存在错译、漏译、逻辑混乱等问题[74]. 本节不涉及依赖参考译文计算 BLEU[86]、METEOR[87]、NIST[88]等译文质量指标的方法, 重点关注能够捕获翻译错误的自动化测试用例生成方法, 而重点关注能够捕获翻译错误的自动化测试用例生成方法, 这些方法根据测试预言构建方式分为基于蜕变测试的方法和基于语料库的方法.

4.2.1 基于蜕变测试的机器翻译测试用例生成方法

由于自然语言的复杂性, 相同语义往往具有多种不同的表达方式. 因此, 机器翻译的正确输出通常存在多种可能, 且无法穷尽列举, 从而导致测试预言难以准确构建. 许多研究者采用蜕变测试理论, 通过定义正确翻译结果应满足的属性来辅助判断翻译正确性. He 等人[66,67]提出了结构不变性、参考透明性这两种蜕变关系, 用于机器翻译测试. 结构不变性是指语义相似的源句的翻译结果应具有相似的语法结构; 参考透明性是指名词短语在不同上下文中应该具有相似的翻译. 上述方法的测试输入生成均基于选定的蜕变关系设计, 例如生成结构相似或包含同一名词短语的一对句子. 与上述方法类似, 机器翻译自动化测试工具 CIT[74]基于成分不变性检测翻译结果. 成分不变性指在添加修饰语后, 句子的语义结构主干不应发生变化, 且添加前后应存在结构路径的包含关系. 满足成分不变性的一对源语言句子, 其翻译结果亦应满足成分不变性. 语义结构主干指句子的基础语法元素所对应的成分解析树, 常见基础元素包括主语、谓语与宾语, 任何一个基础元素的缺失均可能导致语法错误. 修饰语是指句子的可移除部分, 即它的存在不会对句子的主干和语法正确性产生影响. CIT 结合句子压缩模型与 BERT 模型生成具有成分不变性的源语言句子, 其关键技术组件代表了所在领域的先进水平, 能够有效保证测试用例生成的正确性并降低误报率. 与 CIT 通过逐步插入修饰语的方法不同, Zhang 等人[79]提出的 STP 方法利用句法树的基本句子结构与依存关系裁剪输入句子, 以保留核心语义, 再通过词袋模型比较裁剪句子与原始句子的翻译结果一致性. Xie 等人[80]指出, 上述方法在检测翻译错误时, 主要依赖翻译结果之间的整体相似性测量, 粒度过大, 容易导致误报. 因此, 他们提出了基于词闭包的机器翻译测试方法. 该方法首先通过词对齐技术, 关联现有方法生成的源输入、变异输入及对应的翻译输出; 随后, 使用 BERT 模型生成词闭包层面的语义表示, 并计算变异前后具有关联关系的词闭包之间的语义相似度, 以检测翻译错误.

上述方法中蜕变关系的构建主要以翻译结果的语法结构和成分特性为基础, 也有研究者选择更直观的语义维度完成蜕变测试. 此类方法通常定义蜕变关系为: 具有不同语义的源语言句子不应具有相同翻译结果, 而具有相似语义的源语言句子应具有相似翻译结果. 比如, PatInv 通过使用语义不同的词语原词和删除有意义的词语来生成具有不同语义的新句子, 如果新句子的翻译结果和种子句子的翻译结果一致, 则报告翻译错误[68]. Lee 等人[73]使用大小写替换、标点符号改变等针对鲁棒性测试的生成方法来获取语义不改变的新句子. Pesu 等人[69]提出的蒙特卡洛测试方法和 Cao 等人[75] 提出的 SemMT 均采用中间语言生成语义相同的句子. 前者获取直接翻译 (直接从源语言到目的语言) 和间接翻译 (从源语言到中间语言, 再到目的语言) 的翻译结果, 并通过使用 BLEU、Levenshtein

距离和余弦相似度来确定翻译结果之间的差异. 后者使用双向翻译, 即将给定的文本或句子翻译成中间语言 (前向翻译), 然后将结果翻译回源语言 (后向翻译). 之后, SemMT 基于 3 种方式度量翻译结果与原始语句之间的语义相似性, 分别是基于正则表达式的相似性 (regex-based similarity, SemMT-R)、基于确定性有限自动机的相似性 (DFA-based similarity, SemMT-D) 和混合相似性 (hybrid similarity, SemMT-H). 若相似度低于预期阈值, SemMT 则报告一个翻译错误.

此外, 还有研究者将蜕变测试理论应用于机器翻译系统的公平性测试. Sun 等人[78]提出了首个基于蜕变测试的机器翻译系统公平性测试框架——FairMT. FairMT 采用基于语义相似性的蜕变关系, 即语义相近但包含不同人口群体属性的句子, 其翻译结果应具有可比性. FairMT 首先构造涵盖多种人口群体 (demographic groups) 属性 (如性别、国籍和宗教等) 的输入句子, 再基于语义相似度测量识别潜在的不公平翻译, 最终区分真正的公平性问题与由其他类型翻译错误引发的误报. FairMT 着重关注性别与国籍两个属性, 并借助 BiasFinder[89]来生成测试输入. BiasFinder 是一个通过修改人口群体属性来识别情感分析公平性问题的工具, 其中包含的测试输入生成方法符合 FairMT 的功能需求. BiasFinder 是一个通过修改人口群体属性识别情感分析公平性问题的工具, 其包含的测试输入生成方法符合 FairMT 的功能需求.

4.2.2 基于语料库的机器翻译测试用例生成方法

除了基于蜕变关系构造测试预言, 许多研究者还通过人工或自动化方式建立翻译语料库, 以确定正确的翻译结果. 翻译语料库通常具有源文与译文一一对应、精确度高的特点, 因此常用于辅助特定类型的翻译错误检测. 例如, Zheng 等人[70]提出了一种基于源文与译文映射关系的自动化翻译错误识别方法, 主要用于检测过度翻译与欠翻译两类错误. 作者首先爬取了 1 600 万个句子对和短语, 构建源语言与目标语言一一映射的双语语料库, 并利用映射关系检查翻译结果. 欠翻译检测步骤为: 首先获得原文中对应单词/短语的正确翻译列表, 再检查列表内容是否出现在翻译文本中; 过度翻译检测步骤为: 首先统计翻译中每个词语的出现次数, 再通过语料库查找对应源词并统计其在源句中的出现次数, 若源词出现总和小于翻译词出现次数, 则标记为过度翻译. 类似地, Wang 等人[76]提出了一种基于反向推理的机器翻译测试方法 BDTD, 致力于捕获翻译结果中的词义消歧错误. 词义消歧是指确定多义词在给定语境中的确切含义. 作者首先构建并人工审查了多义词的词义库, 再使用性别调换、肯定/否定调换、单复调换及时态调换等变异算子生成语义一致的新句子, 最后通过判断变异前后同一多义词的译文在词义库中的一致性来测试翻译器的词义消歧能力.

还有一些研究者将蜕变测试理论与语料库构建结合, 实现对特定领域翻译的自动化测试. Xu 等人[77]提出了一种针对术语翻译 (terminology translation) 的蜕变测试方法 TermMT, 其核心在于基于术语特征构建蜕变关系: 在原文上下文中添加术语的适当解释说明, 应不影响术语和原文的翻译结果; 修改多词术语的一部分, 应导致翻译结果的相应变化. 为生成能够适配上述蜕变关系的测试输入, TermMT 从现有的数据集 IATE 和 Wikitionary 构建了用于标注和变异的术语词典, 并选择包含单个多词术语的原始句子进行变异. 具体的变异策略包括: 一是含义插入, 即将术语对应的含义以括号形式插入原句; 二是部分替换, 即使用 BERT 模型对术语中的单个单词进行替换. 这两种变异策略分别对应上述两个蜕变关系, 以完成翻译错误的捕获.

此外, 还有一些研究者利用语料库的特性, 完成翻译结果的自动化修正. 这些方法通常都将翻译错误检测作为第 1 步, 因此也会包含测试用例生成. 例如, TransRepair 能够自动检测翻译结果中的不一致性并完成自动化修复[71]. TransRepair 首先使用两个词向量模型构建上下文语义相似语料库, 当两个单词在两个模型中的相似度均超过 0.9 时, 即认为两词上下文相似, 可纳入语料库. 随后, TransRepair 通过将原句中的词语替换为语料库中上下文语义相似的词语生成测试输入, 并采用 TF-IDF、BLEU、LCS 等方法计算翻译结果之间的相似度. 当结果小于设定阈值时, 则认定翻译不一致, 可能存在翻译错误. 然而, Sun 等人[72]认为, TransRepair 中的词语替换方法并不能保证生成句子的正确性, 并提出了新的单词替换方法 CAT 用于生成测试用例. CAT 先使用基于神经网络的语言模型对句子上下文进行编码, 再评估两个词之间的上下文感知语义相似度, 以实现具有可控影响的单词替换. 实验结果表明, 使用 CAT 生成的测试用例相比 TransRepair 具有更高的质量, 能够发现更多数量并覆盖更多类型的翻译错误.

4.3 机器阅读理解

机器阅读理解根据输入类型可分为基于多模态的阅读理解任务[90]和基于文本的阅读理解任务[12], 本节重点关注后者. 基于文本的机器阅读理解任务可以根据问题的回答方式分为 4 类: 完形填空式任务、选择式任务、片段抽取式任务和自由作答式任务. 这 4 种类型任务的目标都是通过理解给定的上下文信息和问题, 选择或者生成一个精简或者复杂的答案. 因此, 机器阅读理解任务的输入通常为一段文本和问题, 输出为问题的答案. 机器阅读理解是自然语言处理领域中一项具有挑战性的任务, 它不仅需要深度理解自然语言, 还需要具备一定的推理能力. 为了多维度评估机器阅读理解系统的性能, 研究者们提出了摆脱传统评估范式限制的测试用例生成方法.

现有的机器阅读理解测试方法通常采用蜕变测试理论去解决测试数据的缺失、对人工标注数据高度依赖等问题. 机器阅读理解任务的输入由一段给定上下文的文本和一个问题构成, 这两部分内容中的任意一个发生语义上的改变, 都会导致正确输出的语义发生相应变化. 为了充分利用现有的人工标注数据 (即数据集中的正确答案集合), 现有研究通常对上下文文本或问题进行变异, 再根据变异后输出的语义是否应发生变化、如何发生变化等相关蜕变关系来确定测试预言. 相比于包含多句话的上下文文本, 输入到机器阅读理解应用的问题在变换过程中更容易做到语义变化程度可控. 因此, 多个研究提出了基于问题变换的机器阅读理解测试方法. 例如, Chen 等人[15]提出了用于自动化验证机器阅读理解软件各项性能的方法, 该方法根据机器阅读理解系统必须遵循的属性制定了两类蜕变关系. 第 1 类是基于反义输出的蜕变关系, 即输出应该与原始输出反义, 并对应 4 种生成新问题的方法: 使用反义形容词、时态改变、时序改变和添加否定词. 第 2 类是基于同义输出的蜕变关系, 即输出应该与原始输出同义, 并对应 3 种生成新问题的方法: 同义形容词替换、状语从句位置改变和语态变化. 基于不同方式产生的测试用例的错误率, 可以相应地估计每个特定属性所对应的自然语言理解能力. 与之类似, Liu 等人[83]设计并实现了一个可用于机器阅读理解任务的模糊测试框架 QATest. QATest 应用语法成分转换、句子结构转换和扰动注入, 对原始种子问题进行变异. 与其他同类型研究工作相比, QATest 的主要特点是采用基于问题数据特点设计的 N-gram 覆盖率和困惑度优先级来指导测试生成过程, 以提高测试效率.

此外, 还有一些研究完全打破了测试用例生成过程中对预标注数据的依赖, 自动生成新的问题或上下文文本. 例如, 基于递归蜕变测试的自动化测试方法 QAASKER, 基于源输入中的问题及被测对象给定的源输出生成一个“伪事实”, 再基于“伪事实”提出一个新问题, 之后将新问题再次输入被测对象得到后续输出[81]. 如果源输出和后续输出都是正确的, 那么源输出与源问题构成的“伪事实”应该等价于后续输出与后续问题构成的“伪事实”; 否则, 源输出和后续输出中至少有一个错误. QAASKER 包含 3 种衍生递归问题的方法: 一是由现有 wh-疑问句的答案衍生出新的 wh-问题; 二是由现有 wh-问题的答案衍生出新的一般疑问句; 三是由现有一般疑问句或选择疑问句的答案衍生出新的 wh-问题. Shen 等人[82]发现 QAASKER 常常产生无效测试用例导致误报, 因此提出了能够生成等价语义测试输入的 QAQA. QAQA 使用了 5 种基于句子级突变的蜕变关系, 并根据所提出的蜕变关系生成等价的问题、等价的上下文、集成的上下文和问题等. 为了更精确地捕获输出错误, QAQA 实现了增强的答案一致性度量作为测试预言. 它使用最先进的短语嵌入模型 PBERT 把输出答案和期望答案表示为向量, 再计算向量之间的余弦相似度作为语义一致性得分. 如果一致性分数小于预定义的阈值, QAQA 则认为违反了蜕变关系, 检测到潜在的缺陷.

4.4 现有方法优缺点分析

为应对自然语言处理应用中输出空间庞大、正确输出难以判断等测试挑战, 现有面向自然语言处理应用的测试用例生成方法大多选用蜕变测试作为核心思路. 现有方法的核心设计主要围绕文本变异策略的选择和测试预言 (蜕变关系) 的构建, 以扩大测试输入空间并高效捕获被测对象的错误输出. 通过总结分析, 本文发现现有方法存在以下待优化的问题: (1) 变异策略与测试预言相互制约: 研究人员选用的文本变异策略主要基于句法结构、语义特征和任务特性, 往往需要在保持或改变输入语义之间进行选择, 而这种选择受到蜕变关系设计与测试目标设定的共同影响, 导致变异策略与蜕变关系之间存在相互制约关系; (2) 输入空间探索性不足: 目前的很多测试输入生成方法还停留在基于局部替换和注入微小扰动的变异策略, 受到种子文本内容制约, 存在输入空间探索有限的

测试瓶颈. 基于上述观察, 表 4 总结了当前面向自然语言处理应用的测试用例生成关键技术路径的优势与不足, 为后续方法设计与改进提供参考.

表 4 现有面向自然语言处理应用的技术路径优缺点总结

技术路径	优点	缺点
基于句法结构的文本变异方法	(1) 可以对句子的句法成分 (如主语、谓语、宾语) 进行精细控制, 从而实现对语义保持或变化程度的有效约束 (2) 生成的句子在句法层面上合法, 有助于生成语法正确、结构清晰的测试输入	(1) 高度依赖句法解析模型的准确性, 在语法结构识别不准确或文本不规范的情况下容易生成无效或错误的测试输入 (2)主要围绕句法结构进行变化, 导致生成的测试用例在表达多样性上存在局限
基于语义特征的文本变异方法	(1) 能深入捕捉语义层面的微妙差异, 适用于检测模型对语义变化的鲁棒性和推理能力 (2) 可结合反义、同义等自然语言现象, 设计更贴近真实语言变异的测试预言	(1) 难以精确控制变异句子的语义偏移程度, 可能导致测试预言模糊或失效 (2) 高度依赖语义词典、语言模型等外部工具, 方法的稳定性和适用性容易受限于资源质量和任务领域
基于任务特性的文本变异方法	(1) 能根据任务输入输出结构设定变异逻辑, 提高测试过程的可控性与效率 (2) 有助于生成更贴合任务需求的测试输入, 便于评估被测对象在关键子任务下的表现	(1) 对具体任务高度依赖, 通用性弱, 难以迁移至其他任务 (2) 需深入理解任务流程与关键语义特征, 设计过程复杂, 增加开发与维护成本
基于语法/语义控制的文本质量保障策略	(1) 能确保测试用例具备良好的语法正确性和语义流畅性, 贴近真实应用场景, 避免语法错误导致误判 (2) 可控制语义变异幅度与公平性, 生成更加可靠、适配性强的高质量测试用例	(1) 文本质量与任务需求可能存在冲突, 过度控制反而降低测试用例的多样性 (2) 需要精细设计控制策略, 不同任务间难以统一标准, 实施难度较大
基于语料库的测试预言构建	(1) 利用人工标注或自动构建的语料库可获得高置信度的参考译文, 测试预言可靠性强 (2) 不依赖模型内部信息, 可用于黑盒测试	(1) 语义差异判断标准难以统一, 有时依赖人为设定的阈值 (2) 语义提取通常依赖外部模型, 可能引入误差干扰
基于句法结构的测试预言构建	(1) 支持覆盖常见错误模式(如遗漏、冗余、歧义), 具备较强的问题定位能力 (2) 借助语料库中标注良好的输入输出对, 能直接构造高精度测试预言	(1) 高质量语料依赖人工标注或数据爬取与清洗, 成本高、领域适应性差 (2) 语料匹配方式通常较静态, 难以应对开放场景下多样化的语言输入
基于语义变化趋势的测试预言构建	(1) 可构建语义变化下的输出预期, 辅助验证模型输出是否随语义调整而改变 (2) 不依赖固定标签, 适用于生成类任务中结果多样性高的场景	(1) 常借助词级替换构造语义变化, 难以刻画真实语义空间中的连续演化 (2) 语义变化前后的输出预期常依赖人工假设, 缺乏统一标准

4.5 小 结

综上所述, 当前面向自然语言处理应用的测试用例生成方法在提升输入多样性与捕获错误行为方面取得了初步成效, 但仍面临变异策略与测试预言耦合度高、测试覆盖范围有限等问题. 因此, 未来的研究方向应着重突破当前由测试预言主导变异策略的局限性, 探索更加灵活、多样化的文本变异方法, 使其能够适配不同类型的测试预言和自然语言处理任务. 具体而言, 可以构建通用的文本变异框架, 确保变异策略适用于不同任务, 提高测试方法的适配性和泛化能力; 引入多样化的文本变异方法, 例如结合大语言模型生成自然、合理的变异文本, 以提升输入数据的多样性和真实性; 提升测试自动化和评估手段, 通过自适应变异、数据驱动方法和多级语义分析, 提高测试覆盖率, 并优化变异策略的有效性, 从而进一步扩大测试输入空间的探索范围, 提高测试的全面性和可靠性.

5 语音处理应用

语音处理通常指将语音信号视为数字信号的一种特例进行处理和研究, 涵盖语音信号的获取、操作、存储、传输和输出等多个方面. 在日常生活中, 常见的语音处理任务包括语音识别、说话人识别等. 在进行分析之前, 语音处理应用首先需要对采集到的语音信号进行语音活性检测, 以确定语音的始末端点, 再实施预加重、分帧、加窗等预处理操作, 以降低人类发声器官和采集设备所带来的混叠、高次谐波失真等因素对语音信号质量的干扰. 之后, 以识别为目标的应用程序会逐帧提取语音特征, 传统特征类型包括 MFCC、PLP、FBANK 等, 并将提取的特征传至解码器进行后续处理操作[91]. 语音识别是语音处理应用的基础, 以语音翻译系统为代表的语音处理应用

的性能与语音识别的精准程度呈显著相关性. 因此, 如表 5 所示, 目前研究者们主要关注面向语音识别任务的自动化测试用例生成方法.

表 5 现有面向语音处理应用的测试用例生成方法总结

发表年份	文献来源	测试输入生成	测试预言构建
2020	[92]	运用TTS模型从文本自动生成语音	生成的语音输入到不同的语音识别系统中, 对比他们的输出结果
2021	[93]		
2022	[94]	(1) 音高变异; (2) 音量变异; (3) 速度变异; (4) 时长变异; (5) 白噪声注入; (6) 环境噪声注入; (7) 麦克风位置修改; (8) 声源位置修改	蜕变关系: 输入包含相同文本序列的语音, 语音识别系统的输出应该相同
2023	[95]	利用ASR系统中的已知识别错误, 从错误文本合成多个语音测试用例	生成的语音输入到不同的语音识别系统中, 对比他们的输出结果
2022	[96]	(1) 噪音注入; (2) 振幅修改; (3) 频率修改; (4) 振幅剪切; (5) 丢帧; (6) 低通滤波器过滤; (7) 高通滤波器过滤	记录了不同群体语音转换前后的识别错误率, 如果两组错误率增量的差值超过了设定阈值, 则存在公平性问题

结合面向语音处理应用的测试用例生成挑战和现有方法的特性, 本文总结出如图 8 所示的面向语音处理应用的测试用例生成研究框架. 现有测试挑战主要包括: (1) 应用环境中的语音输入特征模拟: 语音输入受背景噪声、混响、信道失真、录制设备差异及发音变化 (如语速、音调、口音) 等因素影响, 需要测试系统在不同环境条件和说话方式下的稳健性; (2) 语音语义和质量的可控: 变异后的音频需要保持语义清晰且不失真, 同时控制信号变异幅度, 使其既能触发系统错误, 又符合实际应用场景的合理性; (3) 基于时序特征的测试预言构建: 语音数据的时间动态性会影响被测对象对语音内容的理解, 测试预言需要考虑语音速率变化、语音截断、连续语流及停顿等时序特征, 并实现自动化验证. 为了解决现有挑战, 现有方法从测试输入生成、音频质量控制和测试预言构建这 3 个方面展开. 测试输入生成采用基于文本转换、录音硬件特征及应用环境特征的语音生成方法, 模拟不同设备和噪声条件对语音处理系统的影响. 语音质量控制通过语义约束、变异幅度控制和信号质量评估, 确保测试输入既可能触发错误又不会导致误报. 测试预言构建基于被测对象的输出对比, 找出不一致的具体位置, 或者根据语音变异策略确定参考文本是否改变, 以自动化识别丢词、重复、误识别等错误. 接下来, 本文将更为详细地介绍各类测试用例生成方法.

图 8 面向语音处理应用的测试用例生成方法研究框架

5.1 语音识别

当前, 评估语音识别模型或系统的方法主要是使用现有数据集计算词语错误率 (word error rate, WER)、字符错误率 (character error rate, CER) 等指标[97]. 然而, 通常情况下, 语音识别应用的部署环境包含许多干扰因素, 比如环境噪声、室内回声等. 这些因素可能导致真实应用场景中的实际输入偏离训练数据集分布, 从而使语音识别系统在应用场景中表现出不可预测的行为[94]. 为了解决该问题, 一些研究者提出了基于模糊测试与蜕变测试理论的语音识别测试方法, 设计了多种针对不同应用场景特征的语音转换算子. 这类方法的技术路线可概括为: (1) 使用

语音转换算子生成新语音; (2) 将种子语音和新语音输入被测对象, 获取相应识别结果; (3) 验证新语音与种子语音的识别结果是否满足蜕变关系. 该类方法常用的蜕变关系是: 当输入包含相同文本序列的语音时, 被测对象的输出应保持一致. 该类方法的典型代表为自动化语音识别测试工具 ASRTest, 它集成了 3 组能够模拟真实应用场景中语音的变换算子[94]. 第 1 组是特征变异算子, 即通过改变音高、音量、语速等语音特征来模拟不同人的声音和说话方式, 以评估语音识别系统的说话者适应性. 第 2 组是背景噪声注入算子, 包括白噪声注入与环境噪声注入, 以评估语音识别系统面对各项噪声时的抗干扰能力. 第 3 组是混响模拟算子, 包括麦克风位置修改与声源位置修改, 以评估在不同声学条件下语音识别系统的性能. 与 ASRTest 相类似, 自动化测试框架 AequeVox 使用 8 种与真实应用场景高度相关的转换算子对种子语音进行转换, 包括噪音注入、振幅修改、频率修改、振幅剪切、丢帧、低通滤波器过滤和高通滤波器过滤[96]. 虽然在语音转换方面, ASRTest 和 AequeVox 存在部分相同的转换算子, 但这两个工具测试的侧重点并不相同. ASRTest 致力于以更高的效率捕获语音识别系统的错误行为, 因此集成了基于语音识别系统特性设计的 Gini 不纯度测试用例生成引导算法, 以生成能够触发更多错误的测试用例. 而 AequeVox 致力于度量语音识别系统的公平性, 融合了蜕变测试与差分测试的思想. AequeVox 记录被测对象在输入两组不同群体的语音时的识别错误率以及在实施转换算子后的错误率增量, 若两个群体语音所对应的错误率增量的差值超过了设定阈值, 则认为这违反了公平性.

除此之外, 还有一些语音识别测试方法可归类为差分测试技术, 技术路线可概括为: (1) 基于文本生成测试语音; (2) 将测试语音输入多个被测语音识别系统; (3) 比较多个语音识别系统的识别结果. CrossASR[92]和 CrossASR++[93]是该类型语音识别测试方法的典型代表, 皆集成了不同的文本转语音 (text-to-speech, TTS) 模型, 适配多个语音识别系统, 并包含一个故障评估器. CrossASR 和 CrossASR++的基本工作原理是运用 TTS 模型从文本自动生成语音, 再将这些语音输入到不同的语音识别系统中进行差分测试, 以发现未通过的测试用例. 可以看出, CrossASR 和 CrossASR++生成的测试用例完全依赖于现有文本的质量与数量, 这在一定程度上限制了测试效率. 为解决上述问题, ASDF 在收集一组失败的测试用例后, 使用多种文本转换方法对失败的测试用例进行转换, 以获取更多测试用例生成所需要的文本, 进一步地提升测试用例生成的数量[95]. 例如, 改变一个句子的时态, 或者用其他音素相似的词代替容易出错的词. 此外, ASDF 还包含一个语音分析模块, 该模块通过分析失败的测试用例以找出语音识别系统更倾向于产生错误的音素, 而不是仅汇报失败测试用例的数量. 这 3 种工具具有良好的可扩展性, 用户可选择更先进的 TTS 模型、语音识别系统及更优的故障评估器, 以提升测试性能.

5.2 现有方法优缺点分析

与图像和文本数据相比, 语音具有更强的时序性和声学复杂性, 其测试用例生成面临更多挑战. 目前方法虽已覆盖从 TTS 合成到语音变异的多类技术路径, 但仍存在若干共性不足: (1) 语义一致性保障不足: 生成语音的语义一致性缺乏保障, 尤其在复杂扰动或 TTS 模型精度不足时, 易引发误判, 降低了测试结果的可靠性和参考价值; (2) 测试语音多样性有限: 测试语音的声学多样性与真实环境匹配度仍有限, 难以全面覆盖应用场景, 导致部分特殊场景下的系统鲁棒性评估不足; (3) 错误归因与解释能力薄弱: 测试结果中错误类型的归因与解释能力较弱, 缺乏精细化分析手段, 不利于定位问题根因并指导系统改进. 为进一步分析上述问题的具体表现, 后文表 6 总结了现有面向语音处理应用的测试用例生成技术路径在实践中的典型优缺点.

5.3 小 结

综上所述, 当前面向语音处理应用的测试用例生成方法在模拟真实场景特征与揭示系统鲁棒性方面取得了积极进展, 但仍存在声学变异控制不够精细、语义保持能力有限、测试语音多样性不足等问题. 因此, 未来的研究应进一步突破现有语音变异策略的局限, 探索能够兼顾语音质量与语义一致性的多样化音频生成方法. 具体而言, 可引入结合语音到文本回译机制与音素对齐技术的语义保持手段, 确保语音变异后的内容仍符合原始语义; 构建覆盖多说话人特征、设备差异及环境噪声的系统化测试语音库, 以提升测试的代表性与覆盖广度; 同时, 发展面向不同语音识别系统特性设计的自适应音频变异方法与差异分析机制, 以提高捕获错误输出的针对性与效率, 从而增强面向语音处理应用的测试方法的实用性与泛化能力.

表 6 现有面向语音处理应用的技术路径优缺点总结

技术路径	优点	缺点
基于文本转换的测试音频生成	(1) 通过文本转换生成语音, 可精准控制测试输入的语义特征, 便于构建一致性测试预言 (2) 结合TTS模型可快速批量生成多样性语音输入, 提升测试效率和规模扩展能力	(1) 生成语音在音色、语调等方面受限于TTS模型, 难以覆盖真实环境中的声学多样性 (2) 生成效果高度依赖输入文本质量, 可能导致测试样本分布偏窄或语音内容不自然
基于录音硬件特征的测试音频生成	(1) 可模拟不同麦克风/声源布置对语音输入的影响, 更接近真实部署环境, 提升测试结果的实用性 (2) 有助于发现远场拾音、空间混响等问题下的识别缺陷, 支持部署环境的适配性验证	(1) 受限于现有声学模拟工具的能力, 无法覆盖所有实际声学变化 (2) 固定位置变异策略难以覆盖用户移动、遮挡等复杂互动因素
基于应用环境特征的测试音频生成	(1) 通过注入环境噪声、模拟混响等方式, 测试音频更贴近真实应用环境, 有助于发现语音识别系统在自然场景下的稳定性问题 (2) 可评估系统在多种声学环境下的性能表现(如车内、街道、会议室等), 便于发现场景敏感型缺陷, 提高系统的泛化能力	(1) 如果环境干扰过强或未受控注入, 可能引入语义失真, 进而影响测试预言的准确性, 增加误报风险 (2) 当前缺乏有效的环境建模与验证机制, 无法证明注入的干扰特征确实符合期望模拟的应用场景
基于文本语义变异的语义控制	(1) 通过输入文本的语义调整, 可系统性地生成不同语义层级的语音, 提升测试覆盖率 (2) 根据测试目标灵活选择保持或改变语义, 有助于构造细粒度的蜕变关系或差分关系	(1) 缺乏高效手段去判断TTS输出是否准确传达了预设语义, 可能影响测试可靠性 (2) 语义复杂或结构变化较大的输入文本可能导致TTS生成的语音出现失真的现象
基于语音特征变异幅度的质量控制	(1) 通过调节变异幅度, 实现语音扰动强度的精确控制, 既能触发系统异常, 又能保持语音的可识别性, 确保测试用例的有效性 (2) 可根据测试目的(如鲁棒性测试或边界测试)灵活调整变异强度, 提升测试策略的灵活性和适用范围	(1) 不同被测对象对扰动的敏感度不同, 难以统一界定最能揭示缺陷的变异幅度, 可能需要大量调试 (2) 需要借助语音质量评估模型或指标辅助判断变异语音的可接受性
基于音频信号的质量评估	(1) 直接从音频信号特征出发进行质量评估, 适用于不同语音内容和多语言场景, 具有较强的通用性 (2) 可结合 PESQ、STOI 等音频质量评估指标实现自动化检测, 提升语音测试流程的效率与客观性	(1) 仅依赖信号特征判断质量, 难以判断语音内容是否仍保持原始语义 (2) 某些质量评估指标在特定语速、口音或说话风格下表现不稳定
基于不一致输出的错误捕获	可通过对比多个语音识别系统或同一系统在不同输入下的识别结果, 自动捕获潜在错误, 降低测试人工成本	依赖多个自动语音识别系统对相同音频输入的输出差异进行对比, 若训练数据趋同, 输出结果可能一致, 从而难以有效捕获错误或判断不一致性
基于音频生成/变异策略的预言构建	通过控制生成或变异策略并构建预期的输出一致性关系, 能在不引入人工参考的前提下完成自动测试	音频层面的变异是否保持语义不变在语音识别任务中难以直接确认, 可能导致测试预言不准确

6 点云处理应用

随着三维采集技术的快速发展, 能够提供丰富几何形状和比例信息的三维数据在多个领域得到了广泛应用, 包括自动驾驶、机器人、遥感与医疗[98]. 三维数据通常以不同的格式表示, 如点云、深度图像和体积网格等. 其中, 点云是最为常用的表示形式, 它是分布在三维空间中众多数据点的集合, 数据点是通过激光雷达等传感器来收集. 点云能够在三维空间中完整保留原始几何信息, 且无需离散化处理, 因此成为自动驾驶、机器人等多种场景相关应用的首选表示形式[99]. 根据前文的归纳总结, 本文发现现有面向点云处理任务的测试方法主要围绕自动驾驶任务展开研究. 点云在自动驾驶系统上的应用主要体现在两个方面: (1) 实时环境感知和处理, 用于场景理解和目标检测[100]; (2) 高清地图和城市模型的生成和构建, 用于可靠的定位和参考[101]. 这些应用涉及的核心任务可归纳为 3 类: 三维点云分割、三维目标检测与定位、三维目标分类与识别. 面向自动驾驶领域的点云处理应用, 通常首先需要完成数据接收与读取、运动畸变补偿、点云组帧、外参变化与滤波等预处理操作, 再进行后续的感知与定位步骤[102]. 如表 7 所示, 本文主要关注面向自动驾驶领域中基于点云的目标分割与检测任务的测试用例生成方法.

如图 9 所示, 本文总结了面向点云处理应用的测试用例生成方法的研究框架. 目前, 测试用例生成主要面临以下几个挑战: (1) 点云数据预处理的可控性: 需确保在不同预处理条件下, 点云数据仍能保持一致的几何与语义信息, 以避免因预处理引入额外误差; (2) 点云数据变异的复杂性: 点云数据的变异涉及多种维度, 包括几何形态、密

度及环境因素 (如天气、光照、障碍物等), 且需生成能够覆盖多种现实场景的测试输入; (3) 基于点云语义的测试预言构建: 由于点云数据缺乏结构化的语法规则, 测试预言的构建需结合几何信息、任务特征与环境信息, 以确保自动化评估的有效性. 为应对上述挑战, 现有方法主要通过以下技术路径生成点云数据的测试用例: 基于仿真工具的点云数据合成方法, 通过从虚拟环境中收集点云数据, 确保生成的测试用例涵盖了各种可能的测试场景; 或者, 基于现有点云数据集的方法, 通过利用已有的点云数据集, 进行数据扩展和变异 (如基于仿射变换和环境因素的变异方法), 快速生成多样化的测试用例, 以应对不同测试场景的需求和测试系统在不同实际环境中的稳健性; 最后, 可基于虚拟环境中获取的实例信息或蜕变测试理论, 定义蜕变关系, 以实现错误行为的自动化捕获. 接下来, 本文将按照点云处理任务的分类来进一步地详细讨论分析.

表 7　现有面向点云处理应用的测试用例生成方法总结

发表年份	文献来源	测试输入生成	测试预言构建
2018	[103]	捕获游戏中的画面, 收集对应的点云数据	基于该点的坐标距离, 可获得距离该点的距离、被射线集中的对象类别和实例ID等信息
2022	[104]	(1) 雨天变换; (2) 雪天变换; (3) 雾天变换; (4) 平移; (5) 旋转; (6) 放缩; (7) 对称	蜕变关系: 天气变换后点云的目标检测结果与原始点云的检测结果一致; 仿射变换后点云的目标检测结果与在原始结果上应用仿射算子的结果一致

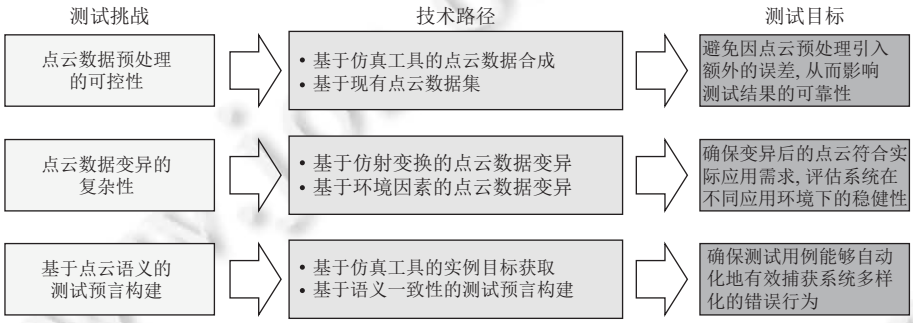

图 9　面向点云处理应用的测试用例生成方法研究框架

6.1　目标分割与检测

当前, 面向自动驾驶领域中基于点云的目标分割和检测任务的测试用例生成方法主要分为两类: 一是从虚拟场景中收集全新的点云数据, 二是对现有的点云数据实施变异. 第 1 种类型的方法不仅需要保证点云数据收集的准确性, 还需要获取到对应目标的相关信息作为测试预言. 例如, Yue 等人[103]提出的点云数据生成框架可实现从计算机游戏“Grand Theft Auto V”中快速创建点云及其对应的精确点级标签, 以生成测试用例. 当游戏中的车辆在虚拟三维街道上行驶时, 该框架可同步采集雷达点云数据并捕获游戏画面. 在数据采集的过程中, 该框架将虚拟激光雷达扫描仪和游戏摄像机置于虚拟三维空间中的相同位置, 这种设置带来了两大优点: 1) 实现对点云数据与图像数据的完整性与一致性检查; 2) 自动完成游戏摄像机与激光雷达扫描仪的配准, 实现点云与图像数据的自动化组合. 该框架使用射线投射 API 来模拟虚拟雷达扫描仪发出的激光, 该 API 可以将射线起始点和结束点的三维坐标作为输入, 并返回射线到达的第 1 个点的三维坐标. 该框架还可基于该点的坐标距离获取其到雷达的距离、被射线命中的对象类别和实例 ID 等信息, 从而实现对收集数据的自动化注释. 该框架生成的点云数据不仅可以用来系统地测试完成点云分割任务的应用, 还可以在训练或是重训练阶段用来提升应用的性能.

第 2 类方法与其他类型智能软件中的蜕变测试方法类似, 通常先使用符合蜕变关系要求的变换算子对点云数据进行变异, 再判断变异前后的目标分割或检测结果是否满足预设蜕变关系. 例如, 用于自动驾驶汽车目标检测任务的自动化测试工具 LiRTest, 定义了基于种子点云数据在不同转换算子作用下目标检测结果之间的蜕变关系, 若违反这些蜕变关系, 则表明存在潜在缺陷[104]. 在测试输入生成方面, LiRTest 实现了两大类变换算子: 天气变换与仿射变换. 天气变换包括模拟雨、雪和雾等环境条件, 仿射变换则包括平移、旋转、缩放与对称操作. 在应用天气

变换算子时, LiRTest 期望转换后点云数据的目标检测结果与原始点云的检测结果一致; 在应用仿射变换算子时, LiRTest 期望转换后点云数据的目标检测结果与在原始结果上应用仿射算子的结果一致. LiRTest 生成的测试用例能够激活目标检测模型的不同神经元, 并通过模拟多种驾驶条件, 有效检测出其错误行为.

6.2 现有方法优缺点分析

点云以三维结构及每点特征对信息进行编码, 具备稀疏性、非结构化和高维度等特点, 使得相关特征提取、变异与语义对齐等操作面临更高难度. 受限于此, 当前面向点云处理应用的测试用例生成方法数量相对较少, 技术成熟度也不高. 尽管部分方法能够通过仿真工具构造数据或利用已有数据集进行变异以扩展输入空间, 但在模拟复杂场景、控制目标语义和获取准确预期结果等方面仍存在诸多问题. 例如, 合成数据的真实性与适配性难以保障, 现有点云数据集的覆盖范围有限, 部分点云变换策略缺乏对语义保持的建模机制, 且多数测试预言依赖于精确标签或映射关系, 难以推广至开放环境中的真实测试. 为系统梳理现有方法存在的关键问题, 表 8 总结了当前面向点云处理应用的主要技术路径及其各自的优势与局限性.

表 8 现有面向点云处理应用的技术路径优缺点总结

技术路径	优点	缺点
基于仿真工具的点云数据合成	(1) 能控制采集过程中的环境变量和对象行为, 生成具备精确语义标签的点云数据, 提高测试的可重复性与可信度 (2) 可在不依赖现实场景的条件下快速生成大规模多样化数据, 扩展测试覆盖范围, 降低人工采集成本	(1) 合成点云与真实场景存在感知差异可能引入模拟偏差, 影响测试结果的外部有效性 (2) 需要仿真平台与游戏引擎的配合, 部署复杂, 会受限于所选仿真工具的建模能力和灵活性
基于现有点云数据集	(1) 利用已有数据可减少采集与合成成本, 便于直接开展测试, 提升测试效率 (2) 许多公开数据集带有精细标注, 适合用于算法验证、基准对比与模型调优	(1) 数据覆盖范围受限, 难以满足多场景、多环境下的稳定性测试需求 (2) 数据集设计目的不一, 预处理方式差异大, 易引入分布偏差影响测试准确性
基于仿射变换的点云数据变异	(1) 可系统性模拟目标在不同空间姿态下的表现, 便于验证被测对象对位姿变化的鲁棒性 (2) 变换方式定义明确、参数可控, 适合构建具有数学可解释性的蜕变关系	(1) 仿射变换未改变点云本身的语义信息, 可能低估模型在复杂真实环境中的潜在缺陷 (2) 仅适用于空间几何变换, 难以覆盖天气、遮挡等更具现实复杂性的测试需求
基于环境因素的点云数据变异	(1) 能模拟多种现实世界复杂环境, 提高测试场景多样性 (2) 有助于评估被测对象在非理想感知条件下的表现, 揭示潜在鲁棒性问题	(1) 环境特征构造缺乏标准化, 难以确保生成点云准确反映目标环境特性 (2) 某些变异可能引入非自然噪声, 干扰被测对象的识别, 导致测试预言难以准确判断
基于仿真工具的实例目标获取	(1) 可自动获取点云的标签与目标实例信息, 显著降低人工标注成本 (2) 具备高度一致性和准确性, 有利于构建高质量的测试预言	(1) 实例信息的真实性依赖仿真环境的建模质量, 存在与现实偏差风险 (2) 仅适用于仿真平台支持的对象类别, 难以覆盖实际场景中的边缘场景
基于语义一致性的测试预言构建	(1) 可利用目标检测或分割结果的一致性构建预言, 规避手工标注需求 (2) 可广泛适配多种输入变异策略, 只要语义保持一致, 便可构建可靠的预言	(1) 难以精确定义“语义一致”的检测结果标准, 存在误判可能 (2) 在复杂场景或多目标情况下, 保持整体语义一致性具有挑战性

6.3 小 结

综上所述, 当前面向点云处理应用的测试用例生成方法在扩展输入空间方面已初见成效, 但仍面临测试数据来源有限、变异策略覆盖不足以及测试预言构建困难等问题. 本文认为, 未来的研究应聚焦以下两个方面, 以进一步弥补当前方法的不足: (1) 研究人员需拓宽点云数据的来源, 探索从更多实际环境中收集数据的可能性, 尤其是针对不常见或难以获取的场景数据; (2) 随着点云数据应用任务的不断多样化, 未来应更加注重设计适配不同应用场景的变异算子, 以确保能够模拟各类环境条件下的点云数据变异. 例如, 在自动驾驶场景中, 点云数据通常还会受到时间与地理位置等因素的影响, 因此需要设计能够适应这些变化的变异算子, 从而提高测试输入空间的多样性, 全面捕获潜在的错误行为. 上述研究内容将有助于推动点云处理领域的测试方法更好地服务于自动驾驶、机器人、遥感等多个行业, 满足这些领域对高效测试的实际需求.

7 多模态数据处理应用

模态被定义为不同的存在形式或信息来源, 例如人类的不同感官 (嗅觉、触觉、视觉等)、信息的载体 (语音、视频、文字等) 以及各种类型的传感器 (雷达、点云、红外线等), 上述每一种都可以被称为一种模态. 由两种或两种以上模态组成的数据称为多模态数据, 如何充分运用多模态数据以提升软件性能, 已成为多个研究领域的重点方向. 自动驾驶系统是当前多模态数据应用成功的典型代表, 其集成的多种传感器能够提供环境及车辆自身的多模态信息. 通过调研发现, 现有面向多模态数据处理应用的测试方法主要围绕自动驾驶系统, 或是用于自动驾驶等任务的感知模块展开. 自动驾驶系统通过融合多模态数据来充分挖掘信息, 提高目标检测和其他组件的性能. 在自动驾驶测试领域, 研究人员通常将被测对象视为黑盒, 通过生成仿真测试场景模拟现实世界中的应用情境, 以全方位评估自动驾驶汽车的安全性、智能性等测试属性[105]. 在测试过程中, 自动驾驶系统可从仿真场景中捕获多模态数据, 作为完成各项任务的基础. 如表 9 所示, 本节重点关注自动驾驶仿真测试中的测试场景生成方法与多传感器融合感知系统的测试用例生成方法.

表 9 现有面向多模态数据处理应用的测试用例生成方法总结

应用领域	发表年份	文献来源	方法类型	测试用例生成
自动驾驶	2018	[106]	交通仿真	利用数据集学习交通行为的基本分布, 并通过模仿学习来训练环境车辆行为的模型和描述人类驾驶策略的模型, 再通过控制标准交通行为的基本分布以估计事故概率, 并将学习到的概率值分配给环境状态和代理行为
	2019	[107]	交通仿真	采用字符串结构和关联规则的语义语言, 分解定义场景的因素, 包括道路、参与者和交通逻辑, 再使用一个基于矩阵的系统以概括场景特征
	2019	[108]	交通仿真	将混合交通流中每个道路使用者的详细行为进行建模, 然后基于每个道路使用者给定的初始状态 (位置和速度), 重建各种道路使用者及其相互作用的复杂行为
	2019	[109]	交通仿真	使用可满足性模理论求解器来自动生成路径, 使用时间自动机的形式开发基于模型检查的动态对象运行生成
	2020	[110]	数字孪生	构造随机感知消息, 包含许多不同类型、大小和位置的静态障碍
	2020	[111]	数字孪生	在模拟器中生成真实世界赛道测试区域的数字孪生, 之后通过使用Scenic语言及VERIFAI工具包在此数字孪生的基础上生成仿真场景
	2021	[112]	交通仿真	通过代码描述具有特定特征的复杂驾驶情况
	2021	[113]	数字孪生	利用几何信息识别地图包含的交叉路口以生成带有交叉路口的抽象场景
	2022	[114]	交通仿真	首先通过组合测试生成抽象场景, 再通过查询地图和模拟器从一个合成的抽象场景中生成一个具体的场景
	2022	[115]	数字孪生	基于UE4引擎和CARLA仿真平台, 生成了总面积10.8 km^2的中国城市风格地图的数字孪生, 并设计了29种模拟交通事件用于生成场景
	2022	[116]	交通仿真	使用场景分布模型来预测无信号交叉口车辆-行人互动的场景的分布
	2022	[117]	交通仿真	基于遗传算法去模拟具有挑战性的切入事件和跟车事件
	2022 2023	[118] [119]	交通法规	使用基于真实世界的交通法规覆盖引导的模糊算法, 通过搜索违反交通法规的不同方式以生成测试场景
	2023	[120]	交通法规	利用遗传算法生成具有多辆自动驾驶车辆的交通控制场景
	2023	[121]	数字孪生	使用带标签的有向图来模拟互通立交的拓扑结构, 并利用 k 路组合覆盖和差分进化生成具体的互通式立交
	2023	[122]	事故报告	使用全景分割模型去从图片和视频中提取真实事故的有效信息, 将不同交通参与者的独立个体分离出来, 为场景恢复提供基础
	2022	[123]	事故报告	使用自然语言处理技术实现现实生活中的文本事故报告到测试场景描述语言的转换
	2022	[124]	交通仿真	从真实交通轨迹中挖掘出有影响力的行为模式以生成安全关键测试场景
	2023	[125]	交通仿真	使用强化学习生成“自杀式”行人的智能体, 以生成涉及行人的安全关键场景
	2024	[126]	交通仿真	运用基于模型的领域特定语言描述场景规范, 涵盖参数、参与者和约束这3部分, 再利用搜索算法生成具体场景
	2024	[127]	事故报告	使用大语言模型从文本车祸报告中提取交互行为, 再转换为逻辑场景, 最后基于搜索算法将逻辑场景转换为具体场景

表 9 现有面向多模态数据处理应用的测试用例生成方法总结 (续)

应用领域	发表年份	文献来源	方法类型	测试用例生成
自动驾驶	2024	[128]	事故报告	使用大语言模型从文本车祸报告中提取道路层、环境层和动态对象层的关键信息, 再结合地图信息针对交互行为完成交通参与者的路径规划
多传感器融合感知	2024	[129]	蜕变测试	采用物理感知的多模态对象插入策略, 从对象数据库选取 3D 对象实例, 经姿态估计确定插入位置和方向, 利用虚拟传感器模拟多传感器数据并处理遮挡, 生成真实且模态一致的测试数据, 运用基于适应度引导的蜕变测试方法, 设计适应度指标评估测试数据揭示错误的能力, 借助蜕变关系自动检测感知系统的错误

由于当前面向多传感器融合感知系统的测试方法较为稀缺, 如图 10 所示, 本文总结了面向自动驾驶系统的仿真测试场景生成研究框架. 当前, 面向自动驾驶系统的仿真测试场景生成主要面对以下挑战: (1) 复杂环境中的多样化模拟与多目标协同: 在自动驾驶系统复杂的应用环境中进行多样化场景与多目标协同行为的模拟具有高度不确定性, 尤其是多目标协同行为之间的合理性与交互关系建模非常困难; (2) 安全攸关场景的仿真数据来源: 当前高质量安全攸关场景数据的获取存在显著困难, 特别是对于那些极端环境或高风险场景, 现有数据往往无法覆盖所有潜在危险情况, 导致高质量仿真数据的收集与生成尤为困难; (3) 自动驾驶系统错误行为的定义与捕获: 由于自动驾驶系统的复杂性特性, 错误行为的定义与捕捉极具挑战性, 因此在测试生成过程中, 必须准确识别不同错误情境, 并通过合适的测试预言进行捕捉, 以确保系统能够有效应对潜在问题. 为解决复杂应用场景难以描述、分析与模拟的问题, 现有方法主要基于领域特定语言实现测试场景建模, 并结合搜索算法等技术手段, 自动生成具有高覆盖率与多样性的测试场景. 部分研究者则使用大语言模型技术, 从仿真数据源中提取有效信息, 并转换为可运行的仿真场景. 仿真数据源方面, 现有研究主要依赖于数字孪生技术、交通事故报告以及交通法规等多种真实数据源. 这些数据源能够有效反映安全攸关场景, 确保测试场景不仅具有高度真实性, 而且能够准确反馈自动驾驶系统在关键安全情境中的表现. 这些数据源也有助于构建测试预言, 例如, 基于交通事故报告提取的场景信息, 可结合碰撞检测、非常规变速等行为模式制定测试预言; 而数字孪生与交通法规生成的场景, 则可以与基于安全标准与合规性验证的测试预言相结合使用. 接下来, 本文将对各个类型的代表性工作进行详细介绍.

图 10 面向自动驾驶系统的测试用例生成方法研究框架

7.1 自动驾驶

7.1.1 基于数字孪生的自动驾驶测试场景生成

自动驾驶系统在正式投入使用之前, 需要在各种复杂场景 (如极端操作条件) 中进行严格测试. 在此过程中, 数字孪生技术可以将物理世界中的场景迁移到计算机仿真软件中, 使工程师能够全面、安全地进行大规模测试. 例如, Fremont 等人[111]提出了一种用于自动驾驶测试的安全性测试场景生成方法. 该方法首先在 LGSVL 模拟器中生成真实世界赛道测试区域的数字孪生, 并通过 Scenic 语言及 VERIFAI 工具包在此基础上生成仿真场景, 进而根据定义的安全指标识别自动驾驶系统在仿真场景中的不安全行为. 此外, 该方法通过度量时序逻辑 (metric temporal

logic, MTL) 计算每个场景的满意度, 并利用主成分分析与聚类技术自动分析安全性与错误分类, 最终通过提取不同的行为模式, 筛选出具有代表性的测试场景. 类似地, Ding 等人[115]提出了可模拟中国交通场景的自动驾驶测试系统. 该系统基于 UE4 引擎与 CARLA 仿真平台, 生成了总面积达 10.8 km^2 的中国城市风格地图的数字孪生. 针对中国最常见的交通事故类型 (如摩托车/电动车事故) 及死亡率最高的事故类型 (如非机动车或行人相关事故), 该系统设计了 29 种模拟交通事件, 用于评估自动驾驶系统的行为. 上述方法均对自动驾驶相关的感知与决策算法提出了更高要求, 有助于缩小仿真场景与现实世界之间的差距.

除了直接利用物理世界的数字孪生生成测试场景外, 还有部分研究进一步分析与处理已有数字孪生数据, 先找出其中的关键场景, 再生成对应的仿真场景, 以提升测试效率. 例如, 自动化方法 SALVO 能够以地图的数字孪生为基础, 利用几何信息识别地图中的交叉路口, 进而生成带有交叉路口特征的抽象场景[113]. 为提升测试效率, SALVO 采用了基于距离度量与特征映射的贪婪算法, 筛选相似场景, 并通过在车道上放置静态障碍物进行场景扩增. 最后, SALVO 将选定的抽象场景实例化到工业级驾驶模拟器中, 并自动化执行生成的场景. 该方法能够从不同复杂地图中识别出大量与交叉路口相关的关键驾驶场景, 所生成的测试用例可有效暴露自动驾驶系统中不易察觉但安全性关键的问题. 此外, 还有一些研究聚焦于利用数字孪生技术建模特殊的道路场景, 而不再需要建模的完整路面地图. 例如, 模型驱动的测试方法 FLYOVER 针对当前缺乏公路互通式立交测试场景的问题, 使用带标签的有向图模拟互通立交拓扑结构, 并参考现实世界中的立交数据以保证拓扑的实用性, 进而对拓扑模型进行分类, 提取不同的拓扑等价类[121]. 对于每个拓扑类别, FLYOVER 能够识别匝道的几何特征, 并通过 k 路组合覆盖与差分进化方法, 生成具体的互通式立交场景. FLYOVER 能够用于生成由不同互通式立交组成、具有可测量多样性的测试数据集.

7.1.2 基于交通事故报告的自动驾驶测试场景生成

基于数字孪生的自动驾驶测试场景致力于还原真实世界的环境, 但在某些情况下, 测试自动驾驶系统并不需要完整的环境信息, 从事故信息中提取关键道路场景进行还原将更具效率与意义. 基于交通事故报告的测试场景生成方法首先需要从事故报告中提取场景关键信息, 再将这些信息转换为仿真场景中的关键组成部分, 最终补全缺失部分形成完整可运行的测试场景. 由于事故报告具有不同的载体形式, 相应的数据提取与处理技术也各不相同. 例如, 全景分割模型 M-CPS (multi-channel panoptic segmentation) 用于从图片和视频中提取真实事故的有效信息, 并将不同交通参与者的独立个体分离出来, 为场景恢复提供基础[122]. 相比于传统的分割模型, M-CPS 充分考虑了事故现场图片和视频的特征, 提升了小物体检测的准确性, 更有效地提取现场信息. 为了进一步提升信息提取的准确性, 作者采用了与文本描述交叉验证的方案, 以消除不必要的交通参与者, 并捕获核心交通参与者的运动轨迹, 最终在自动驾驶开放平台 Apollo 和仿真器 CARLA 上建立了测试场景集.

此外, 自动驾驶测试平台 ADEPT 使用自然语言处理技术, 将现实生活中的文本事故报告转换为测试场景描述语言[123]. ADEPT 预设了一组与测试场景相关的问题和对应的 Scenic 代码模板, 将事故报告文本与预定义问题输入 GPT-3 模型, 再根据模型回答, 从模板库中选择与描述相对应的模板, 并填充缺失信息. 具体来说, 问题查询的内容包括碰撞的位置 (三叉路、四叉路或者非交叉路口)、时间、天气以及两辆车之间的相对位置关系 (包括车道的左右关系、距离的前后关系等). 随着大语言模型 (large language model, LLM) 的发展, 研究人员越来越多地将 LLM 技术应用于基于文本车祸报告的仿真测试场景生成. 例如, SoVAR 通过多轮提示设计, 直接从文本车祸报告中提取道路层、环境层与动态对象层的关键信息[128]. SoVAR 的研究重点在于仿真场景中交通参与者的路径规划. SoVAR 将给定地图解析为一个包含详细道路类型记录的候选道路集合, 遍历其中的道路, 当当前道路与事故发生道路类型匹配且交通参与者占用的最大车道数不超过所选道路车道数时, 调整参与者行驶方向与初始车道位置以完成匹配. 最后, SoVAR 依据对参与者驾驶动作以及驾驶动作之间的约束条件, 求解生成一系列路点, 从而完成仿真场景中的轨迹规划. 与 SoVAR 不同, LeGEND 以文本车祸报告为输入, 通过设计两个提示语, 借助 LLM 先提取事故参与者的关键信息和交互模式, 再将其转换为领域特定语言 (DSL) 表示的逻辑场景[127]. 基于生成的逻辑场景, LeGEND 使用多目标遗传算法搜索关键具体场景, 以场景关键性、交互性与多样性为目标函数, 引导搜索方向. 基于交通事故报告的场景生成方法为构建场景集提供了新的方向, 上述方法的实验结果表明, 该类方法更有助

于捕获自动驾驶系统的性能边界.

7.1.3 基于交通仿真的自动驾驶测试场景生成

在自动驾驶的测试场景中, 除了道路信息, 行人、其他车辆等参与者的行为也是交通场景中的重要组成部分. 因此, 许多研究者以参与者行为为切入点, 研究基于交通仿真的自动驾驶测试场景生成方法, 该类方法更关注参与者对自动驾驶系统可能产生的影响. 基于交通仿真的自动驾驶测试场景生成方法通常通过分析现有交通数据, 提取多种参与者的单独行为模式及参与者之间的交互模式, 并将其应用于仿真测试, 以生成更贴近真实应用场景的测试用例. 例如, O'Kelly 等人[106]提出的罕见交通事件模拟工具利用美国交通部公共交通数据建模以学习交通行为的基本分布, 并通过模仿学习来训练环境车辆行为的模型和一组描述人类驾驶策略的模型. 该方法通过控制标准交通行为的基本分布估计事故概率, 结合自适应重要性抽样方法与交叉熵算法, 迭代近似最优重要性采样分布, 从而加速罕见事件的概率评估, 并将学习到的概率值分配给环境状态和代理行为. 与之类似, 自动驾驶测试工具 CRISCO 通过从真实交通轨迹中挖掘出有影响力的行为模式, 即一些常见的导致交通事故的参与者的行为[124]. CRISCO 通过检测参与者轨迹是否发生重叠来判断是否发生碰撞, 确定碰撞轨迹并进行分类, 最终获取到 8 种汽车的行为模式和两种行人的行为模式. 之后, CRISCO 构建简单抽象的场景, 通过求解轨迹规范生成多个具体测试场景, 并逐步引导参与者实施有影响力的行为, 以提升非关键场景的挑战性. 这类方法能够快速有效地生成使自动驾驶系统发生崩溃的安全关键场景.

此外, 还有一些研究为解决现有的模拟器很少模拟自动驾驶车辆与其他交通参与者之间相互作用的问题, 提出了多种模拟车辆与行人、车辆与自行车等交互模式的方法. 例如, Song 等人[116]提出了一个专注于无信号灯交叉路口建模的场景分布模型, 该模型采用交通密度和碰撞时间 (time to collision, TTC) 来预测车辆-行人交互的场景分布. 该方法使用泊松分布来决定在给定时间范围内出现一定数量的车辆或行人的概率, 将行人进入交叉口的瞬间前方车辆撞到行人之前的剩余时间定义为碰撞发生的紧急程度来评估自动驾驶系统的性能. Chao 等人[108]提出的基于力的概念来实现异构交通仿真的方法可以通过简单统一的方式准确地复制各种道路使用者的复杂行为及其相互作用. 在构建模拟环境的过程中, 该方法主要关注影响自动驾驶汽车决策的交互作用, 即车辆与行人的交互作用和车辆与自行车的交互作用. 作者将混合交通流中每个参与者的详细行为进行建模, 然后基于参与者的初始状态 (位置和速度) 去重建各种参与者及其相互作用的复杂行为. Yang 等人[125]提出了一种在 CARLA 模拟器中设计"自杀式"行人智能体的方法, 用于测试自动驾驶系统在涉及行人的危险情况下的安全性. 该行人被建模为一个强化学习智能体, 并配备了两个自定义奖励函数, 使其不按照常规交通规则或行人行为模式行动, 而是倾向于与自动驾驶汽车发生碰撞. 实验结果表明, "自杀式" 行人在识别自动驾驶系统的决策错误方面是有效的, 但生成的仿真场景的真实性与合理性有待进一步确认.

目前, 将搜索优化算法与交通仿真相结合去实现更高效率地找到符合预设目标的关键交通仿真场景也是一个值得探索的方向. 其中, 一些研究使用遗传算法生成关键交通仿真场景, 其主要原理是通过构造适应度函数, 对编码后的参数进行搜索, 直到达到研究人员所设目标的最优解或是最大迭代次数. 例如, Zhou 等人[117]提出了基于遗传算法的切入事件和跟车事件的模拟框架, 该框架由两个阶段组成: 初始场景生成和场景库生成. 在初始场景生成阶段, 每个场景被视为一个个体, 个体的适应度 (TTC 指数) 代表了对场景的挑战性评估, 该框架对原始种群的个体基因随机初始化, 并选择基于适应度的轮盘赌选择方法来确定初始场景. 在场景库生成阶段, 对于给定的初始场景, 由该场景作为起点进行水平搜索和垂直搜索, 以找到具有挑战性的场景. 若找到, 算法将其更新为新的初始场景, 由该场景作为起点重新进行搜索. 除了将生成具有挑战性的场景作为搜索或优化目标, 一些研究将场景多样化作为最终求解目标, 例如 Kim 等人[109]提出的可以从路径和行为规范自动化生成仿真测试场景的测试用例生成框架和 Li 等人[114]提出的场景覆盖率驱动的自动驾驶安全测试工具 ComOpT. 前者使用了一种可满足性模理论 (satisfiability modulo theories, SMT) 求解器来自动生成路径, 并引入了测试区域覆盖的概念, 根据生成的路径可以覆盖的区域的多样性来量化生成路径的质量. ComOpT 通过组合测试生成抽象场景, 并使用 k 路组合测试作为覆盖标准, 其目标是确保测试场景集能够覆盖任意 k 个类别的所有组合. ComOpT 将抽象场景生成问题定义为混合整数线性规划问题以最大限度地增加覆盖率, 每个抽象场景都可以被视为概念等价类. 之后, ComOpT 通过查询地

图和模拟器从抽象场景中生成一个具体的场景, 包括找到满足所有地理条件的子地图, 车辆定位, 配置 NPC 车辆和行人, 以及设置天气等模拟参数. 该类方法的特征是在一定程度上对方法生成的场景完成了质量评估, 更有利于高效率地完成自动驾驶系统的仿真测试.

除此之外, 还有一些研究者致力于研究如何用更利于测试人员理解和使用的方式去创建交通场景, 常采用的方法是以类结构化语言的形式去灵活地定义场景. 例如, Medrano-Berumen 等人[107]提出的用于自动驾驶测试的抽象仿真场景生成框架. 作者开发了一种采用字符串结构和关联规则的语言, 用于分解定义场景的因素, 包括道路、参与者和交通逻辑. 基于此语言, 该框架可获取到单个路段的描述或不同类型角色 (汽车、卡车、行人等) 的行为描述, 并将这些描述解析以生成场景. 与之相比, Paracosm 更注重系统化地探索自动驾驶系统的状态空间, 包括视觉特征和与环境的反应性交互[112]. Paracosm 允许用户以编写代码的方式来描述具有特定特征的复杂驾驶情况, 例如道路布局和环境条件, 以及其他车辆和行人的反应性时间等. 为了最大限度地覆盖到被测对象的各种行为, Paracosm 使用了基于组合测试和拟蒙特卡洛方法基本思想的测试覆盖率, 并在参数配置时结合了模糊测试技术. 该类方法使用类似于代码编程的方式对参数化的测试环境进行建模, 能够实现高效率地生成可以捕获自动驾驶系统缺陷的仿真测试场景. 与 Paracosm 类似, Concretize 允许用户运用基于模型驱动的领域特定语言描述场景规范, 涵盖参数、参与者和约束这 3 部分[126]. 不同的是, Concretize 利用元启发式搜索或完全搜索方法将抽象规范转化为具体场景, 再对生成的场景进行可视化展示、模拟运行, 监测分析被测对象的行为, 并生成测试结果的图表.

7.1.4 基于交通法规的自动驾驶测试场景生成

为了确保道路安全, 自动驾驶汽车的行为必须满足一系列复杂的规则, 例如现有的交通法规以及未来可能针对自动驾驶汽车的法律. 为了全面测试自动驾驶系统, 研究人员希望明确被测系统是否可能会违反某些交通法规, 提出了一些基于交通法规的自动驾驶测试场景生成方法. 这些方法通常是以现有的交通法规为自动驾驶系统的行为判断标准, 探索如何生成会使自动驾驶系统违反交规的测试场景. 这些场景可以揭露出被测对象可能存在的各个方面的安全性隐患. 例如, 可兼容不同场景描述语言的 LawBreaker 提供了面向驾驶员的特定领域语言来描述交通法规, 并包含了一个模糊引擎, 其实现了基于法规覆盖引导的模糊算法, 可通过最大化覆盖范围来搜索违反交通法规的不同方式[118]. 以 LawBreaker 为基础, 自动驾驶测试方法 ABLE 使用 LawBreaker 生成初始场景, 引入了具有动态评分函数和主动学习的 GFlowNet 方法, 基于信号时序逻辑去实现动态更新测试目标, 再结合主动学习来驱动测试过程, 以有效地探索广泛的搜索空间, 从而高效地生成多种违反交规的测试场景[119]. 实验表明, ABLE 生成的场景比 LawBreaker 生成的场景更加有效, 能使自动驾驶系统出现更多种违反交通法规的方式. 此外, 还有一些研究结合了遗传算法去生成可能使自动驾驶系统违反安全和交通规则的场景. 例如, DoppelTest 利用遗传算法生成具有多辆自动驾驶车辆的交通控制场景, 来捕获自动驾驶系统的违规行为[120]. 在该遗传算法中, 每个场景被视作个体, 其基因由自动驾驶车辆, 行人以及交通控制配置组成. 这 3 类基因是彼此独立的, 可分别进行修改. DoppelTest 通过选择、突变和交叉这 3 种方式对基因进行修改. 在每一代中, DoppelTest 使用适应度对个体进行评价, 以最小化场景对象间距离、最大化决策数量、最大化冲突轨迹和最大化违规次数为目标. 在进行违规分析时, DoppelTest 选择了 4 类测试预言来评估自动驾驶系统, 包括碰撞违规, 交通信号违规, 停车标志违规以及模块响应失败.

7.2 多传感器融合感知

多传感器融合感知系统通过整合不同传感器的数据, 提升智能系统的感知精度和可靠性. 传统的单一传感器存在局限性, 例如相机能提供丰富的语义信息但易受环境影响, 而激光雷达 (LiDAR) 可获取高精度的三维几何数据但缺乏语义理解, 因此相机-激光雷达融合成为常见方案. 基于融合阶段, 多传感器融合技术可分为早期融合、晚期融合、深度融合和弱融合, 其中深度融合通过跨模态特征交互提升感知效果. 随着人工智能技术的迅猛发展, 多传感器融合感知系统在自动驾驶、机器人等众多前沿领域的目标检测等关键任务中发挥着不可替代的关键作用. 因此, 现有测试用例生成方法也是基于目标检测的任务特征去设计.

目前, 面向该类系统的测试用例生成仍然面临诸多挑战: (1) 跨模态一致性的问题: 与单传感器感知系统不同, 多传感器融合感知系统需合成跨不同传感器的模态一致测试用例. 例如, 同一物体在图像和相应点云中的姿态应保持一致. 然而, 现有的基于图像和点云的测试方法简单组合无法保证这种一致性. (2) 语义真实性的问题: 在变异种子测试用例时, 若没有适当的物理约束, 极大概率会出现不合理的情况. 比如, 插入的物体可能出现在无效位置, 或者导致错误的视角关系, 如被遮挡但仍可见的建筑物. 这些不真实的测试用例无法有效检测感知系统在实际场景中的性能.

为解决上述挑战, Gao 等人[129]提出了 MultiTest, 一种基于适应度引导的蜕变测试方法, 专门用于多传感器融合感知系统的测试. MultiTest 包含 3 个主要步骤: 姿态估计、物理感知多传感器模拟和适应度引导测试. 在测试开始时, 给定一个多模态测试种子 (即一对图像帧和点云帧), MultiTest 会自动从数据库中选择一个 3D 目标实例, 然后将其插入到原始数据中. 为了生成语义合理的测试数据, MultiTest 首先会搜索插入目标的有效姿态, 再使用物理感知传感器模拟模块合成逼真的图像和点云帧. 这一过程包含激光雷达模拟、相机模拟和遮挡处理. 在激光雷达模拟中, MultiTest 模拟激光束发射并考虑噪声影响; 在相机模拟中, 利用 Blender 开源软件构建虚拟相机, 并借助 S^2CRNet 模型优化插入对象的颜色, 使其与场景融合自然. 同时, MultiTest 还处理了插入对象与背景数据之间的遮挡问题, 确保数据的真实性. MultiTest 定义了蜕变关系来创建测试预言, 判断对象插入前后系统预测结果是否满足特定条件, 从而自动检测感知结果中的错误, 避免了繁琐的手动标注. 此外, 为了提升测试效率, MultiTest 设计了一种适应度引导的测试过程. 通过计算包含对象漏检、误检和定位错误等因素的适应度指标, 优先保留具有高错误揭示能力的测试用例, 使得生成的测试集能够更有效地检测系统错误. 实验结果表明, MultiTest 方法能有效检测被测系统的错误行为, 并通过再训练提高系统的鲁棒性.

7.3 现有方法优缺点分析

经过调研分析, 本文发现面向多模态数据处理应用的测试用例生成方法已在学术界和工业界引起广泛关注, 尤其是在自动驾驶等典型场景中, 不同模态传感器数据的融合带来了更高的感知能力和系统鲁棒性. 但由于传感器之间的数据形式差异大、语义融合复杂、测试任务多样, 现有方法在场景生成与测试预言构建方面仍面临诸多挑战: (1) 测试场景语义信息提取的准确性和一致性不足: 当前研究多依赖图像、点云、文本等模态进行融合建模, 但由于各类模态中语义提取技术仍存在局限, 导致融合生成的测试场景在关键要素识别、行为重建等方面存在信息缺失或语义冲突, 影响最终测试效果; (2) 方法的适配性与可复用性不足: 多模态测试往往依赖特定平台、工具或仿真环境, 许多方法缺乏跨平台部署能力, 通用性有限, 难以支撑规模化、系统化的测试实践. 为更全面地理解当前面向多模态数据处理应用的测试研究现状, 表 10 对已有主流技术路径的优缺点进行了系统性归纳与总结.

表 10 现有面向多模态数据处理应用的技术路径优缺点总结

技术路径	优点	缺点
基于搜索算法的交通协同场景生成	(1) 可系统性探索大规模参数空间, 高效生成具有挑战性的关键测试场景 (2) 支持优化多种目标函数 (如碰撞风险、多样性), 生成场景具备更强测试覆盖能力	(1) 搜索过程依赖目标函数设计, 适应性和泛化能力受限于目标定义的合理性 (2) 在真实地图或高保真模拟环境中搜索存在时间和计算成本压力, 效率受限
基于大语言模型的场景描述解析	(1) 能从自然语言中提取丰富场景信息, 支持从事故报告等文本生成可复现场景 (2) 可结合prompt工程等技术实现语义补全与细节推理, 提升场景还原度	(1) 场景语义解析结果依赖大语言模型, 存在理解偏差与生成不一致的问题 (2) 缺乏对解析结果合理性与完整性的系统验证机制, 影响测试场景的可靠性
基于领域特定语言的测试场景建模	(1) 提供结构化、形式化的场景建模方式, 便于精确控制交通元素与行为逻辑 (2) 可复用已有语法规则与模型库, 高效支持复杂交通场景的快速构造与调整	(1) 语言设计需结合具体平台和测试需求, 通用性受限, 迁移成本较高 (2) 场景建模灵活性依赖语言表达能力, 难以覆盖非规则或极端复杂场景
基于数字孪生的仿真场景生成	(1) 能高度还原真实世界环境, 增强测试场景的真实性和代表性 (2) 可结合地图拓扑与场景行为特征, 系统挖掘高风险或关键场景	(1) 数字孪生构建依赖高质量地图和传感数据, 成本高、流程复杂 (2) 不同地区或环境的建模规范不统一, 难以跨平台共享与复用

表 10 现有面向多模态数据处理应用的技术路径优缺点总结 (续)

技术路径	优点	缺点
基于交通事故报告的仿真场景生成	(1) 能从真实事故中提取高风险行为模式, 生成具有挑战性的关键测试场景 (2) 可提升测试效率, 聚焦自动驾驶系统的安全边界与性能瓶颈	(1) 事故报告内容格式多样, 信息提取与结构化过程依赖强大解析能力 (2) 语义理解和要素补全仍存在不确定性, 生成场景的完整性和一致性难以保障
基于交通法规的仿真场景生成/测试预言构建	(1) 明确以交通法规为基准, 可系统性地发现潜在违规行为和安全缺陷 (2) 测试预言清晰、具备解释性, 便于对被测系统的判断逻辑进行验证	(1) 交通法规复杂多变, 场景抽象与规则映射过程存在不确定性 (2) 需要对法规进行形式化建模, 构建与模拟器兼容的约束机制, 开发成本高
基于行为模式识别的测试预言构建	(1) 能有效发现具有潜在危险性的异常行为模式, 提升对边缘场景的检测能力 (2) 基于轨迹或交互模式的判断方式可自动化构建测试预言, 减少人工标注负担	(1) 行为模式的抽象和分类依赖大量真实轨迹数据, 初始建模成本较高 (2) 不同系统对行为模式的反应差异较大, 测试预言的通用性和一致性存疑
基于安全性标准的测试预言构建	(1) 基于安全指标 (如碰撞、有无违规、TTC等) 构建测试预言, 可以直接衡量系统是否违反基本安全约束, 评判结果更明确 (2) 安全性标准通常是跨平台的约束要求, 不依赖于具体系统的实现细节, 适合广泛的系统对比和统一评估	(1) 不同任务或系统下合理的TTC、车距等阈值可能不同, 统一设定可能带来误判或漏判风险 (2) 无法识别非违反规则但存在隐患的行为模式, 难以全面评估系统的实际风险水平

7.4 小 结

综上所述, 当前面向多模态数据处理应用的测试用例生成方法在自动驾驶等关键领域取得了积极进展, 但仍面临场景语义提取和生成不准确、测试用例生成方法通用性不足等问题. 针对上述问题, 未来的研究可围绕以下几个重点方向展开, 以提升自动驾驶测试场景的生成质量和测试方法的通用性. (1) 基于国际标准的场景描述语言进行测试场景生成: 未来研究可以依托符合国际标准的自动驾驶测试场景描述语言 OpenSCENARIO, 探索更加灵活、高效的场景生成方法. OpenSCENARIO 提供了一种结构化的方式来描述复杂的驾驶场景, 使其可以适配多种自动驾驶仿真软件, 如 CARLA、LGSVL 等. 通过利用 OpenSCENARIO, 可在跨平台适配性和测试覆盖面方面取得更大突破, 提高研究成果的工程可落地性. (2) 跨模态数据融合提升场景语义提取能力: 结合多模态数据融合的方法, 提高场景语义信息提取的准确性. 例如, 利用计算机视觉技术解析事故现场图像, 结合激光雷达点云信息, 还原真实事故场景, 并通过自然语言处理技术解析对应的文本报告, 提高生成的测试用例的真实性和有效性.

8 深度学习模型

除了上述面向某个应用领域中具体任务的测试用例生成方法以外, 目前也有许多研究聚焦于面向智能软件系统的核心组件——深度学习模型的测试用例生成方法. 数据驱动的深度学习模型通过学习一组训练数据来构造内部逻辑, 因此这类方法在设计时重点考虑与模型内部逻辑相关的深度神经网络结构特征和数据流动, 比如神经元状态与输出、层与层之间的连接等. 本文根据欲解决的实际研究问题, 将这些相关方法分为测试用例度量方法和测试用例生成方法进行归纳总结.

8.1 测试用例度量方法

由于被测对象的完整输入空间无法穷尽覆盖, 现有动态软件测试方法通常围绕测试用例设计展开, 旨在通过有限测试用例触发的软件行为, 尽可能有效地反映整体软件质量. 测试用例对应的反映程度被定义为测试充分性. 测试用例充分性度量用于评估测试用例集或单个测试用例在覆盖被测对象行为方面的充分性. 传统软件测试方法中的测试度量体系包含了语句覆盖、判断覆盖和条件覆盖等, 但这些方法并不能直接应用于深度学习模型测试. 如表 11 所示, 受传统覆盖指标启发, 研究者们提出了多种面向深度学习模型测试的覆盖度量指标. 这些方法通过分析模型内部神经元的激活值、状态变化及网络结构特性, 一方面支持面向测试用例集的粗粒度充分性评估, 另

一方面也实现了针对单个测试用例揭示模型错误行为的细粒度度量. 目前, 许多面向深度学习模型的测试用例度量指标都是受到传统软件测试的基本概念和方法的启发而被提出. 在这些指标中, 应用范围最为广泛的是用于深度学习测试的神经元覆盖率[130]. 类似于传统软件测试中的代码覆盖率, 神经元覆盖率被定义为输入测试用例后, 处于激活状态的神经元数量占神经网络总神经元数量的比例. Pei 等人[130]指出, 一组测试用例如果能够尽可能多地激活神经元, 即神经元覆盖率越大, 则该测试集对于深度学习模型而言越全面, 也更有可能包含导致模型出错的极端案例. 然而, 也有研究者指出, 简单地计算激活神经元比例无法充分反映深度学习模型的结构特性. 为此, 提出了面向深度学习系统的多粒度测试指标——DeepGauge[131]. DeepGauge 从单个神经元和层级两个粒度计算神经元覆盖率, 分别是神经元级别的 k-多区域神经元覆盖率、神经元边界覆盖率和强神经元激活覆盖率, 以及层级的 top-k 神经元覆盖率. 相比于单维度的神经元覆盖率, DeepGauge 中多粒度的神经元覆盖率计算方式能够更好地评估测试用例对深度学习系统可靠性的检测能力.

表 11 现有面向深度学习模型的测试用例度量指标总结

方法来源	发表年份	度量指标
DeepXplore[130]	2017	神经元覆盖率
DeepGauge[131]	2018	神经元级: k-多区域神经元覆盖率、神经元边界覆盖率、强神经元激活覆盖率 层级: top-k神经元覆盖率
DeepCruiser[132]	2018	状态级: 基本状态覆盖率、k-step状态边界覆盖率 转换级: 基础转换覆盖率、输入空间覆盖率、加权输入覆盖率
DeepCT[133]	2019	t-way组合稀疏覆盖率、t-way组合密集覆盖率、(p, t)-完整性覆盖率
Structural test coverage[134]	2019	符号-符号覆盖、距离-符号覆盖、符号-值覆盖、距离-值覆盖
SADL[135,136]	2023	意外充分性、意外覆盖率
DeepImportance[137]	2020	重要性驱动覆盖率
DeepGini[85]	2020	Gini不纯度
Test selection for deep learning systems[138]	2021	模型不确定性
Diversity-guided method[139]	2022	蜕变测试用例对的多样性

除了代码覆盖率外, 一些研究者受修正条件判定覆盖率 (modified condition/decision coverage, MC/DC) 启发, 提出了针对深度神经网络结构与语义特征的 4 类覆盖率评估指标. MC/DC 的核心思想是要求测试覆盖所有与决策相关的条件变化. 在深度神经网络中, 可以通过不同方式定义决策受条件变化影响的机制. Sun 等人[134,140]基于神经元符号、数值或距离变化特征, 提出了 4 种捕捉测试输入因果变化的测试度量指标: 符号-符号覆盖、距离-符号覆盖、符号-值覆盖以及距离-值覆盖. 此外, 传统软件测试中的组合测试思想也被引入到深度学习测试领域. 传统的组合测试的核心思想是从系统输入的所有取值组合中选择一个小规模的子集作为测试用例集合. 该集合使得对于任意 t 个参数的所有可能取值的组合被至少一个测试用例覆盖. 受此启发, Ma 等人[133]提出了 t-way 组合稀疏覆盖率、t-way 组合密集覆盖率以及 (p, t)-完整性覆盖率.

然而, 一些研究者认为, 直接调整传统软件工程中的测试指标以适配深度学习模型, 难以有效捕获深度学习模型的内在特性. 为解决这一问题, Gerasimou 等人[137]提出了系统性测试方法 DeepImportance, 并引入了重要性驱动覆盖率 (importance-driven coverage, IDC). DeepImportance 首先分析深度学习模型的输入、输出及内部神经元行为, 明确内部神经元对输出结果的贡献关系. 随后, 它识别出与模型行为因果关联性更强的重要神经元, 并将每个重要神经元的行为空间划分为自动确定的有限集群. DeepImportance 利用 IDC 度量测试用例集合的充分性, 即衡量该集合覆盖的重要神经元行为集群组合的比例.

上述指标主要适用于测试用例集合, 属于粗粒度的度量方式. 而测试框架 SADL 则支持更细粒度地度量单个测试用例的测试充分性. SADL 包含用于衡量深度学习模型在面对测试输入时相对于训练数据的意外程度的两个指标, 即意外充分性 (surprise adequacy, SA) 和意外覆盖率 (surprise coverage, SC)[135,136]. SA 有两种计算方式, 可细分为基于核密度 (kernel density estimation, KSE) 计算的 LSA 和基于欧几里得距离计算的 DSA. SC 用于确定测试

用例集合的意外充分性范围, 因此同样存在两种对应的计算方式. SADL 通过对比测试用例和训练数据的 SA 值, 来确定该测试用例相对于训练数据的意外程度.

前文所述的测试度量指标主要是面向 CNN 网络, 并不适用于存在回路及网络内部状态转移的 RNN 网络. 因此, Du 等人[132]将 RNN 抽象为马尔可夫决策过程模型, 以刻画 RNN 的内部状态与动态行为. 在此基础上, 提出了一套针对有状态深度学习系统的测试度量体系, 包含 2 个状态级测试覆盖准则和 3 个转换级测试覆盖准则. 状态级测试覆盖指标分别是基本状态覆盖率和 *k*-step 状态边界覆盖率, 前者主要比较测试输入访问的抽象状态数量与训练阶段的抽象状态数量, 后者主要考察测试输入触发训练阶段从未访问过的抽象状态的情况. 转换级测试覆盖准则针对的是由不同输入触发的时间动态行为, 分别是比较测试和训练阶段抽象转换的基础转换覆盖、为保证后续转换多样性的输入空间覆盖和考虑后续转换范围的加权输入覆盖.

由于深度神经网络模型的特性, 基于模型内部神经元输出及状态的测试度量计算通常耗时较长, 可能导致测试效率下降. Ma 等人[138]将模型返回的类别预测概率视为评估测试输入挑战性程度的依据, 并据此提出了基于概率的模型不确定性度量指标. 实验结果表明, 与覆盖率度量指标相比, 基于不确定性度量的指标在识别导致分类错误的输入方面表现更优. 且其计算开销更低, 在模型重训练期间也能更快速地提升分类准确性. 类似地, DeepGini 仅依赖于 Softmax 层的输出即可计算 Gini 不纯度, 用于预测输入引发分类错误的可能性[85]. DeepGini 不纯度的设计原理是, 当模型对各类别输出的概率相近时, 测试输入发生分类错误的概率较高. 与需要完整深度神经网络内部信息的覆盖指标相比, DeepGini 仅依赖于输出层向量信息, 因此具有更高的测试效率和更强的可扩展性. 由于测试用例通常具有不同的输出向量, 但可能拥有相同的神经元覆盖率, 因此 Gini 不纯度相比神经元覆盖率具有更好的区分能力.

尽管已有多种衡量测试用例充分性的指标被提出, 但这些指标难以有效评估蜕变测试用例对 (metamorphic test case pair, MP) 的多样性, 因而难以进一步明确测试充分性的提升方向. 基于神经元覆盖率的指标主要考虑几个固定的神经元激活状态, 无法细粒度地描述深度神经网络的内部执行状态. 为此, Xie 等人[139]提出了用于度量 MP 多样性的指标, 以识别缺陷敏感的 MP, 并采用更细粒度的方法对神经元输出差异建模, 从而获得更精确的内部执行状态表示. 该度量将 MP 中两个测试用例的神经元输出之间的分布差异作为该 MP 的多样性. 两个测试用例具有较大神经元输出分布差异的 MP 可能覆盖更多不同的深度神经网络执行模式, 例如依赖于更多不同特征的路径, 从而更容易揭示潜在缺陷. 该度量指标不仅能够有效且稳定地促进深度学习测试中缺陷信息的揭示, 还能辅助选择和设计更具效能的蜕变关系.

测试用例度量指标常会与测试用例选择方法相结合, 研究者们已将传统的基于覆盖的测试用例选择技术中的基于总覆盖率的选择方法 (coverage-total method, CTM) 和基于附加覆盖率的选择方法 (coverage-additional method, CAM) 用于深度学习模型测试, 实现测试用例优先级排序[141]. CTM 总是优先选择覆盖率最高的测试用例, 其次是覆盖率第 2 高的测试用例, 以此类推. 对于具有相同覆盖率的测试用例, CTM 将随机对测试用例进行优先级排序. CTM 的优点在于仅需根据测试用例的覆盖率进行排序, 因而高效且易于实现. CAM 与 CTM 的不同之处在于, CAM 在选择下一个测试用例时, 会根据已有选择结果的反馈信息进行决策. 它迭代地选择一个可以覆盖更多未覆盖情况的测试用例, 以提升测试场景的多样性. CAM 相比于 CTM 会更加耗时, 每次选择时都需要重新计算剩余未选择测试的覆盖率信息. 除此之外, 也有一些研究者针对深度学习模型测试常用的蜕变测试理论设计了测试用例选择方法. 例如, Arreita[142]提出了一种基于多目标搜索的测试用例选择方法用于深度学习系统的蜕变测试. 蜕变测试需要对多个测试用例 (种子测试用例和基于它生成的新测试用例) 的输入和输出进行断言, 因此会更加耗时. 为了解决该问题, 该方法通过选择置信度低的对应种子测试用例来选出具有高概率违反蜕变关系的新测试用例.

8.2 测试用例生成方法

由于本文不涉及对抗性样本生成, 所以下文所提到的方法都是将测试目标定为最大化选定的模型度量指标, 或者使用特定的方法对种子测试用例进行变换去模拟深度学习模型可能会处理到的输入数据, 以扩大测试覆盖的

输入空间. 其中大多数方法的输入数据类型是图片, 种子数据集包括 MNIST[143]、CIFAR-10[144]和 ImageNet[145]等, 因此涉及的种子变换方法都是以第 3 节中讨论的图片变换方法为主.

DeepHunter[146]和 DeepConcolic[147]都是选择基于覆盖的度量指标作为测试用例评估标准, 将最大化覆盖率指标设为测试用例生成的引导方向. DeepHunter 的完整测试流程包括: 首先通过种子选择策略筛选合适种子, 随后应用蜕变变异策略生成新的语义保留测试用例, 并以覆盖率指标作为反馈, 指导测试生成及新测试用例是否进一步突变的决策. 目前, DeepHunter 已经集成了 5 种基于覆盖率的度量指标和 4 种种子选择策略. 与使用模糊测试方法的 DeepHunter 不同, DeepConcolic 将 Concolic 测试技术应用到深度学习系统测试中. Concolic 测试结合程序执行与符号分析, 以探索软件程序的执行路径, 其中符号分析能够引导程序执行, 从而以更少的执行次数发现潜在错误. DeepConcolic 通过基于启发式的具体执行和符号执行, 增量地生成高覆盖率的测试用例. 相比于同样使用神经元覆盖率的 DeepXplore[130], DeepConcolic 能够在更短的时间内达到更高的覆盖率, 具有更高的测试性能.

此外, 研究者们也将搜索算法应用到面向深度学习模型的测试用例生成中, 例如 DeepJanus[148]和 DeepAtash[149]. DeepJanus 针对 DNN 的行为边界实现了一个多目标进化算法, 通过产生真实的输入对被测对象的行为边界进行彻底探索. 深度学习的行为边界是一组彼此相似并触发深度学习系统不同行为的输入对, 表示深度学习系统可以按预期正常输出的输入区域的边界. 一个低质量的深度学习系统的行为边界可能包括有效性域相交的输入对, 而高质量的深度学习系统更偏向于在面对大幅度偏离的输入时表现异常, 与有效域的交集很小或没有. 与 DeepJanus 不同, DeepAtash 旨在解决开发人员需要获取应用场景中新特征数据以微调 DNN 模型、达到所需精度水平的问题. DeepAtash 支持配置不同的搜索策略 (单目标或多目标) 和稀疏度指标. 它将所需的目标特征值范围作为输入, 并把生成输入的稀疏度和生成输入与目标在特征图中的接近程度作为优化目标. 最终, DeepAtash 生成具有预定义特征的能够诱导被测对象产生错误行为的输入.

8.3 小 结

目前, 大多数的面向深度学习模型的测试度量指标和测试用例生成方法都是以神经元覆盖的思想为核心. 但已有研究表明, 该类型的度量指标与模型质量并不一定相关, 尤其是无法体现出模型的鲁棒性、公平性等质量属性[150]. 未来, 研究人员可以着手于进一步明确覆盖度量指标与测试用例多样性之间的关系, 重点探究能够反映模型质量属性的度量指标. 同时, 在保证效果的前提下, 应致力于将度量指标的计算过程轻量化, 以简化深度学习模型的测试流程, 降低其在工程应用场景中的适配难度. 上述研究路线可作为当前研究的补充, 不仅能够量化深度学习模型的缺陷, 还能够为在不同维度上提升模型质量提供依据和设计思路. 此外, 现有一些研究, 比如 Aries[150]、PACE[151]和 CES[152], 聚焦于深度学习模型的性能预估, 主要目的是减少数据标注成本. 尽管它们的研究重点并不是如何度量测试用例, 但将此类技术与细粒度测试用例选择相结合仍具有潜力, 未来探索这一方向将有助于提升测试效率与精准度.

9 未来研究方向

虽然已经有许多研究聚焦于面向智能软件系统的测试用例生成, 但当前仍然存在许多不足与挑战. 本文立足于现有研究成果, 系统概括了当前面临的主要挑战, 并对未来的研究方向进行了展望, 旨在为后续工作提供思路, 助力智能软件系统质量的持续提升.

9.1 面向智能软件系统的多模态测试用例生成

经过系统地梳理, 本文发现当前面向智能软件系统的测试用例生成方法大多适配于文本和图像这两种数据类型, 面向语音、点云等存储形式和处理方式更为复杂的数据类型的测试用例生成方法较少. 近年来, 研究人员致力于将不同模态的数据进行整合, 以让智能软件系统获得更丰富、全面和准确的信息. 除了典型的自动驾驶场景, 当前多模态数据的热门应用还包括: 结合文本、语音和视频数据进行情感分析与识别; 结合医学影像、病历文本和生理参数进行病情评估、病灶检测与患者风险预测; 结合视频监控、传感器数据和交通事件报告实现交通拥堵检测、事故预警和路径规划等任务. 因此, 面向智能软件系统的多模态测试数据生成具有较大的探索价值和空间.

当前多模态测试用例生成方法主要关注于覆盖不同模态的组合状态, 但在生成多样化测试用例方面的研究仍然存在不足. 未来的研究可以探索如何通过引入更多的变异算子来增加生成测试用例的多样性, 以更全面地覆盖多模态数据的不同情况和特征. 其次, 多模态数据通常包含异构模态, 未来研究可以关注如何有效地生成针对异构模态数据的测试用例, 这通常涉及如何处理和整合不同模态数据的特征表示、语义关联以及模态间的转换和映射等问题. 此外, 多模态数据在各个领域得到广泛应用, 未来研究可以关注如何生成针对不同应用场景的多模态测试用例, 以更好地模拟实际应用中的数据特征和需求, 提高测试用例的真实性.

9.2 面向智能软件系统的测试用例生成效率优化

目前, 许多智能软件系统测试方法把智能软件系统看成黑盒, 其中的测试用例生成方法属于数据驱动, 与模型内部无关. 数据驱动测试将数据从逻辑脚本中分离出来, 并通过调整参数来更新测试数据以增加灵活性. 数据驱动测试可以高效地构建测试数据集, 满足智能软件系统测试的大规模数据需求. 然而, 与生成的庞大测试数据集相比, 真正能够捕获错误的测试用例比例较小, 导致大部分测试时间被消耗在运行未能揭示错误的测试用例上. 因此, 如何在测试用例的生成过程中去引导生成能够较大概率揭示问题的测试用例, 或者基于测试用例的输入去预测系统的行为, 将是提升智能软件测试效率的关键突破口.

经过观察与分析, 本文发现, 一些基于数据驱动的测试方法由于未利用系统核心模型的结构等内部特性, 难以高效生成能够发现错误的测试用例, 从而降低了整体测试效率. 现有的一些灰盒测试方法结合了基于模型内部输出的度量指标去对生成过程实现引导, 并取得了不错的效果提升. 从中可以看出, 结合模型内部输出或者结构特性的测试方法是切实可行的, 研究者们可以在不同类型的智能软件上进行更多的尝试. 此外, 如何与提高模型行为可解释性的相关技术有效结合, 以提升测试用例的生成效率, 也可能是提升智能软件测试用例生成方法效率的重要切入点.

9.3 面向智能软件系统的测试用例生成迭代优化

当前在智能软件测试方法中, 错误行为被定义为错误的输出, 通常可被表示为代表输入输出的数据对, 具体的表现形式与被测试的任务紧密相关. 当前的测试用例生成方法大多聚焦于如何找出智能软件系统的错误行为, 而对如何利用捕获到的错误行为数据进行错误溯源、因果分析等反馈机制的研究较少. 这导致现有方法难以基于测试用例的执行结果进行迭代优化, 在后续生成过程中也无法有针对性地产生高质量测试用例. 上述研究内容能够最大化测试用例执行结果的利用率, 不仅能够提升智能软件测试的效果和效率, 对于后续的缺陷修复也能够提供更多有价值的信息.

智能软件系统的错误溯源是指通过分析系统的错误行为来追溯和定位错误产生的原因和过程. 未来的研究可以利用数据挖掘和机器学习等技术, 对测试用例捕获的错误日志等信息实施系统化分析与挖掘, 以发现错误产生的模式和规律. 分类、聚类和关联规则算法等也可以用来分析错误行为的特征和属性, 以识别常见的错误类型和错误相关因素. 基于错误溯源的结果, 测试用例生成方法可以在下一轮生成过程中, 产生与错误原因相对应的测试用例以验证溯源结果, 或生成符合错误模式或规律的测试输入以捕获类似缺陷. 此外, 还可以设计具有不同特征和属性的新测试输入, 以探索尚未覆盖的输入空间. 未来的研究还可以进一步地设计相应的系统质量改进措施, 包括代码修复、数据清洗、模型调优等技术手段, 以改善智能软件系统的可靠性、稳定性和一致性.

10 总 结

目前, 研究者们基于不同的技术路线提出了面向各类智能软件系统的测试用例生成方法, 以不断完善智能软件系统的质量评价体系. 本文将智能软件系统按照人工智能技术的应用领域进行初步划分, 再按照系统实现的任务目标进行归纳. 具体来说, 本文介绍了面向图像处理应用、自然语言处理应用、语音处理应用、点云处理应用、多模态数据处理应用以及深度学习模型的各类测试用例生成方法, 形成全面详细的综述. 在此基础上, 本文进一步讨论了当前面向智能软件系统的测试用例生成方法的不足之处以及未来的研究方向. 随着智能软件系统在众多领域中的广泛应用, 它已成为当前工业界和学界重点关注的研究与应用焦点. 针对智能软件系统复杂的决策逻辑和

动态环境适应性, 设计合理、效果优异的测试用例生成方法对于系统质量保障至关重要. 因此, 面向智能软件系统的测试用例生成方法不仅具有良好的工程应用前景, 还在理论研究上展现出巨大的发展潜力, 预计将在未来的软件工程等领域持续引领新的研究热点.

References

[1] Apple Inc. Apple's Siri. 2025. https://www.apple.com/siri/

[2] Amazon.com, Inc. or its affiliates. Amazon's cloud-based voice service. 2025. https://developer.amazon.com/en-US/alexa

[3] OpenAI. ChatGPT. 2025. https://chat.openai.com/

[4] Baidu. Baidu's wenxin. 2025. https://yiyan.baidu.com/

[5] Waymo LLC. Waymo. 2025. https://waymo.com/intl/zh-cn/

[6] General Motors. Cruise self driving cars. 2025. https://getcruise.com/

[7] Huang CX, Cai HN, Xu LD, Xu BY, Gu YZ, Jiang LH. Data-driven ontology generation and evolution towards intelligent service in manufacturing systems. Future Generation Computer Systems, 2019, 101: 197–207. [doi: 10.1016/j.future.2019.05.075]

[8] TED. Amazon alexa and devices division on pace to lose $10b. 2022. https://www.strata-gee.com/amazon-alexa-and-devices-division-on-pace-to-lose-10b-div-in-crisis-mode/

[9] Smith L. Israel police mistakenly arrest Palestinian man for writing 'good morning' on Facebook. 2017. https://www.independent.co.uk/news/uk/home-news/israel-police-palestinian-man-arrest-good-morning-facebook-page-translation-mistake-a8015626.html

[10] Thailand has threatened to take legal action against Facebook. 2020 (in Chinese). https://k.sina.cn/article_1949671172_74359f0400100rsmq.html

[11] Ribeiro MT, Wu TS, Guestrin C, Singh S. Beyond accuracy: Behavioral testing of NLP models with checklist. arXiv:2005.04118, 2020.

[12] Rajpurkar P, Jia RB, Liang P. Know what you don't know: Unanswerable questions for SQuAD. arXiv:1806.03822, 2018.

[13] Recht B, Roelofs R, Schmidt L, Shankar V. Do ImageNet classifiers generalize to ImageNet? In: Proc. of the 36th Int'l Conf. on Machine Learning. Long Beach: PMLR, 2019. 5389–5400.

[14] Wu TS, Ribeiro MT, Heer J, Weld D. Errudite: Scalable, reproducible, and testable error analysis. In: Proc. of the 57th Annual Meeting of the Association for Computational Linguistics. Florence: ACL, 2019. 747–763. [doi: 10.18653/v1/P19-1073]

[15] Chen SQ, Jin S, Xie XY. Validation on machine reading comprehension software without annotated labels: A property-based method. In: Proc. of the 29th ACM Joint Meeting European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 590–602. [doi: 10.1145/3468264.3468569]

[16] Zhang JM, Harman M, Ma L, Liu Y. Machine learning testing: Survey, landscapes and horizons. IEEE Trans. on Software Engineering, 2022, 48(1): 1–36. [doi: 10.1109/TSE.2019.2962027]

[17] Xiang WM, Musau P, Wild AA, Lopez DM, Hamilton N, Yang XD, Rosenfeld J, Johnson TT. Verification for machine learning, autonomy, and neural networks survey. arXiv:1810.01989, 2019.

[18] Huang XW, Kroening D, Ruan WJ, Sharp J, Sun YC, Thamo E, Wu M, Yi XP. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review, 2020, 37: 100270. [doi: 10.1016/j.cosrev.2020.100270]

[19] Braiek HB, Khomh F. On testing machine learning programs. Journal of Systems and Software, 2020, 164: 110542. [doi: 10.1016/j.jss.2020.110542]

[20] Wang Z, Yan M, Liu S, Chen JJ, Zhang DD, Wu Z, Chen X. Survey on testing of deep neural networks. Ruan Jian Xue Bao/Journal of Software, 2020, 31(5): 1255–1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5951.htm [doi: 10.13328/j.cnki.jos.005951]

[21] Vishnukumar HJ, Butting B, Müller C, Sax E. Machine learning and deep neural network—Artificial intelligence core for lab and real-world test and validation for ADAS and autonomous vehicles: AI for efficient and quality test and validation. In: Proc. of the 2017 Intelligent Systems Conf. (IntelliSys). London: IEEE, 2017. 714–721. [doi: 10.1109/IntelliSys.2017.8324372]

[22] Pannu A. Artificial intelligence and its application in different areas. Artificial Intelligence, 2015, 4(10): 79–84.

[23] Dabre R, Chu CH, Kunchukuttan A. A survey of multilingual neural machine translation. ACM Computing Surveys, 2020, 53(5): 99. [doi: 10.1145/3406095]

[24] Malik M, Malik MK, Mehmood K, Makhdoom I. Automatic speech recognition: A survey. Multimedia Tools and Applications, 2021, 80(6): 9411–9457. [doi: 10.1007/s11042-020-10073-7]

[25] Adamopoulou E, Moussiades L. An overview of chatbot technology. In: Proc. of the 16th IFIP WG 12.5 Int'l Conf. on Artificial

Intelligence Applications and Innovations. Neos Marmaras: Springer, 2020. 373–383. [doi: 10.1007/978-3-030-49186-4_31]

[26] Yurtsever E, Lambert J, Carballo A, Takeda K. A survey of autonomous driving: Common practices and emerging technologies. IEEE Access, 2020, 8: 58443–58469. [doi: 10.1109/ACCESS.2020.2983149]

[27] Peters BS, Armijo PR, Krause C, Choudhury SA, Oleynikov D. Review of emerging surgical robotic technology. Surgical Endoscopy, 2018, 32(4): 1636–1655. [doi: 10.1007/s00464-018-6079-2]

[28] Ye SJ, Zhang PC, Ji SH, Dai QY, Yuan TH, Ren B. Survey on non-functional attributes for AI-enabled software systems and quality assurance methods. Ruan Jian Xue Bao/Journal of Software, 2023, 34(1): 103–129 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6409.htm [doi: 10.13328/j.cnki.jos.006409]

[29] Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science, 2006, 313(5786): 504–507. [doi: 10.1126/science.1127647]

[30] LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521(7553): 436–444. [doi: 10.1038/nature14539]

[31] Li ZW, Liu F, Yang WJ, Peng SH, Zhou J. A survey of convolutional neural networks: Analysis, applications, and prospects. IEEE Trans. on Neural Networks and Learning Systems, 2022, 33(12): 6999–7019. [doi: 10.1109/TNNLS.2021.3084827]

[32] Salehinejad H, Sankar S, Barfett J, Colak E, Valaee S. Recent advances in recurrent neural networks. arXiv:1801.01078, 2018.

[33] van Houdt G, Mosquera C, Nápoles G. A review on the long short-term memory model. Artificial Intelligence Review, 2020, 53(8): 5929–5955. [doi: 10.1007/s10462-020-09838-1]

[34] Larochelle H, Bengio Y, Louradour J, Lamblin P. Exploring strategies for training deep neural networks. The Journal of Machine Learning Research, 2009, 10: 1–40.

[35] Liu B, Wei Y, Zhang Y, Yang Q. Deep neural networks for high dimension, low sample size data. In: Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence. Melbourne: AAAI Press, 2017. 2287–2293.

[36] Wenzel J, Matter H, Schmidt F. Predictive multitask deep neural network models for ADME-Tox properties: Learning from large data sets. Journal of Chemical Information and Modeling, 2019, 59(3): 1253–1268. [doi: 10.1021/acs.jcim.8b00785]

[37] Tesla autopilot crashes again: Does not slow down directly into the white van. 2020 (in Chinese). https://nev.ofweek.com/2020-06/ART-71005-8440-30442524.html

[38] Takanen A, Demott JD, Miller C. Fuzzing for Software Security Testing and Quality Assurance. Norwood: Artech House, 2018.

[39] Chen TY, Cheung SC, Yiu SM. Metamorphic testing: A new approach for generating next test cases. arXiv:2002.12543, 2020.

[40] McKeeman WM. Differential testing for software. Digital Technical Journal, 1998, 10(1): 100–107.

[41] Feng D, Haase-Schütz C, Rosenbaum L, Hertlein H, Gläser C, Timm F, Wiesbeck W, Dietmayer K. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. IEEE Trans. on Intelligent Transportation Systems, 2021, 22(3): 1341–1360. [doi: 10.1109/TITS.2020.2972974]

[42] Yuan XY, He P, Zhu QL, Li XL. Adversarial examples: Attacks and defenses for deep learning. IEEE Trans. on Neural Networks and Learning Systems, 2019, 30(9): 2805–2824. [doi: 10.1109/TNNLS.2018.2886017]

[43] Ilyas A, Santurkar S, Tsipras D, Engstrom L, Tran B, Madry A. Adversarial examples are not bugs, they are features. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 125–136.

[44] Hagler DJ Jr, Hatton S, Cornejo MD, *et al*. Image processing and analysis methods for the adolescent brain cognitive development study. NeuroImage, 2019, 202: 116091. [doi: 10.1016/j.neuroimage.2019.116091]

[45] Ren XL, Li XY, Ren KJ, Song JQ, Xu ZC, Deng KF, Wang X. Deep learning-based weather prediction: A survey. Big Data Research, 2021, 23: 100178. [doi: 10.1016/j.bdr.2020.100178]

[46] Maier A, Syben C, Lasser T, Riess C. A gentle introduction to deep learning in medical image processing. Zeitschrift für Medizinische Physik, 2019, 29(2): 86–101. [doi: 10.1016/j.zemedi.2018.12.003]

[47] Czimmermann T, Ciuti G, Milazzo M, Chiurazzi M, Roccella S, Oddo CM, Dario P. Visual-based defect detection and classification approaches for industrial applications—A survey. Sensors, 2020, 20(5): 1459. [doi: 10.3390/s20051459]

[48] Jalled F, Voronkov I. Object detection using image processing. arXiv:1611.07791, 2016.

[49] Wang P, Zhang ZY, Zhou YQ, Huang ZQ. Test data augmentation for image recognition software. In: Proc. of the 20th IEEE Int'l Conf. on Software Quality, Reliability and Security Companion (QRS-C). Macao: IEEE, 2020. 280–284. [doi: 10.1109/QRS-C51114.2020.00054]

[50] Tian YC, Zhong ZY, Ordonez V, Kaiser G, Ray B. Testing DNN image classifiers for confusion & bias errors. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 1122–1134. [doi: 10.1145/3377811.3380400]

[51] Dreossi T, Ghosh S, Sangiovanni-Vincentelli A, Seshia SA. Systematic testing of convolutional neural networks for autonomous driving. arXiv:1708.03309, 2017.

[52] Tian YC, Pei KX, Jana S, Ray B. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In: Proc. of the 40th Int'l Conf. on Software Engineering. Gothenburg: ACM, 2018. 303–314. [doi: 10.1145/3180155.3180220]

[53] Wang S, Su ZD. Metamorphic object insertion for testing object detection systems. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2021. 1053–1065. [doi: 10.1145/3324884.3416584]

[54] Shao JY. Testing object detection for autonomous driving systems via 3D reconstruction. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. (ICSE-Companion). Madrid: IEEE, 2021. 117–119. [doi: 10.1109/ICSE-Companion52605.2021.00052]

[55] Wang XL, Yang SQ, Shao JY, Chang J, Gao G, Li M, Xuan JF. Object removal for testing object detection in autonomous vehicle systems. In: Proc. of the 21st IEEE Int'l Conf. on Software Quality, Reliability and Security Companion (QRS-C). Haikou: IEEE, 2021. 543–549. [doi: 10.1109/qrs-c55045.2021.00083]

[56] Shorten C, Khoshgoftaar TM. A survey on image data augmentation for deep learning. Journal of Big Data, 2019, 6(1): 60. [doi: 10.1186/s40537-019-0197-0]

[57] Dasiopoulou S, Mezaris V, Kompatsiaris I, Papastathis VK, Strintzis MG. Knowledge-assisted semantic video object detection. IEEE Trans. on Circuits and Systems for Video Technology, 2005, 15(10): 1210–1224 (in Chinese with English abstract). [doi: 10.1109/TCSVT.2005.854238]

[58] Yuan L, Li XM, Pan ZX, Sun JM, Xiao L. Review of adversarial examples for object detection. Journal of Image and Graphics, 27(10): 2873–2896. [doi: 10.11834/jig.210209]

[59] Liu L, Ouyang WL, Wang XG, Fieguth P, Chen J, Liu XW, Pietikäinen M. Deep learning for generic object detection: A survey. Int'l Journal of Computer Vision, 2020, 128(2): 261–318. [doi: 10.1007/s11263-019-01247-4]

[60] Xie C, Wang JY, Zhang ZS, Zhou YY, Xie LX, Yuille A. Adversarial examples for semantic segmentation and object detection. In: Proc. of the 2017 IEEE Int'l Conf. on Computer Vision (ICCV). Venice: IEEE, 2017. 1378–1387. [doi: 10.1109/ICCV.2017.153]

[61] Otter DW, Medina JR, Kalita JK. A survey of the usages of deep learning for natural language processing. IEEE Trans. on Neural Networks and Learning Systems, 2021, 32(2): 604–624. [doi: 10.1109/TNNLS.2020.2979670]

[62] Liu ZX, Feng Y, Chen ZY. DialTest: Automated testing for recurrent-neural-network-driven dialogue systems. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2021. 115–126. [doi: 10.1145/3460319.3464829]

[63] Shen XC, Chen HB, Chen JF, Zhang JW, Wang SH. EcoDialTest: Adaptive mutation schedule for automated dialogue systems testing. In: Proc. of the 2023 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Taipa: IEEE, 2023. 933–939. [doi: 10.1109/SANER56733.2023.00113]

[64] Chen HB, Chen JF, Wu YC, Cai SH, Ahmad B, Huang RB, Wang SR, Zhang C. DialTest-EA: An enhanced fuzzing approach with energy adjustment for dialogue systems via metamorphic testing. Software Testing, Verification and Reliability, 2025, 35(1): e1897. [doi: 10.1002/stvr.1897]

[65] Guo GX, Aleti A, Neelofar N, Tantithamthavorn C. MORTAR: Metamorphic multi-turn testing for LLM-based dialogue systems. arXiv:2412.15557, 2024.

[66] He PJ, Meister C, Su ZD. Structure-invariant testing for machine translation. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 961–973. [doi: 10.1145/3377811.3380339]

[67] He PJ, Meister C, Su ZD. Testing machine translation via referential transparency. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 410–422. [doi: 10.1109/ICSE43902.2021.00047]

[68] Gupta S, He PJ, Meister C, Su ZD. Machine translation testing via pathological invariance. In: Proc. of the 28th ACM Joint Meeting European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 863–875. [doi: 10.1145/3368089.3409756]

[69] Pesu D, Zhou ZQ, Zhen JF, Towey D. A Monte Carlo method for metamorphic testing of machine translation services. In: Proc. of the 3rd Int'l Workshop on Metamorphic Testing. Gothenburg: ACM, 2018. 38–45. [doi: 10.1145/3193977.3193980]

[70] Zheng WJ, Wang WY, Liu D, Zhang CR, Zeng QS, Deng YT, Yang W, He PJ, Xie T. Testing untestable neural machine translation: An industrial case. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. (ICSE-companion). Montreal: IEEE, 2019. 314–315. [doi: 10.1109/ICSE-Companion.2019.00131]

[71] Sun ZY, Zhang JM, Harman M, Papadakis M, Zhang L. Automatic testing and improvement of machine translation. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 974–985. [doi: 10.1145/3377811.3380420]

[72] Sun ZY, Zhang JM, Xiong YF, Harman M, Papadakis M, Zhang L. Improving machine translation systems via isotopic replacement. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1181–1192. [doi: 10.1145/3510003.3510206]

[73] Lee DTS, Zhou ZQ, Tse TH. Metamorphic robustness testing of Google translate. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on

Software Engineering Workshops. Seoul: ACM, 2020. 388–395. [doi: 10.1145/3387940.3391484]

[74] Ji P, Feng Y, Liu J, Zhao ZH, Xu BW. Automated testing for machine translation via constituency invariance. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 468–479. [doi: 10.1109/ASE51524.2021.9678715]

[75] Cao JL, Li MZN, Li YT, Wen M, Cheung SC, Chen HM. SemMT: A semantic-based testing approach for machine translation systems. ACM Trans. on Software Engineering and Methodology, 2022, 31(2): 34e. [doi: 10.1145/3490488]

[76] Wang J, Li YH, Huang X, Chen L, Zhang XF, Zhou YM. Back deduction based testing for word sense disambiguation ability of machine translation systems. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 601–613. [doi: 10.1145/3597926.3598081]

[77] Xu YH, Li YH, Wang J, Zhang XF. Evaluating terminology translation in machine translation systems via metamorphic testing. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 758–769. [doi: 10.1145/3691620.3695069]

[78] Sun ZY, Chen ZP, Zhang J, Hao D. Fairness testing of machine translation systems. ACM Trans. on Software Engineering and Methodology, 2024, 33(6): 156. [doi: 10.1145/3664608]

[79] Zhang QJ, Zhai J, Fang CR, Liu JW, Sun WS, Hu HC, Wang QY. Machine translation testing via syntactic tree pruning. ACM Trans. on Software Engineering and Methodology, 2024, 33(5): 125. [doi: 10.1145/3640329]

[80] Xie XY, Jin S, Chen SQ, Cheung SC. Word closure-based metamorphic testing for machine translation. ACM Trans. on Software Engineering and Methodology, 2024, 33(8): 203. [doi: 10.1145/3675396]

[81] Chen SQ, Jin S, Xie XY. Testing your question answering software via asking recursively. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 104–116. [doi: 10.1109/ASE51524.2021.9678670]

[82] Shen QC, Chen JJ, Zhang JM, Wang HY, Liu S, Tian MH. Natural test generation for precise testing of question answering software. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2023. 71. [doi: 10.1145/3551349.3556953]

[83] Liu ZX, Feng Y, Yin YN, Sun JY, Chen ZY, Xu BW. QATest: A uniform fuzzing framework for question answering systems. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2023. 81. [doi: 10.1145/3551349.3556929]

[84] Kann K, Ebrahimi A, Koh J, Dudy S, Roncone A. Open-domain dialogue generation: What we can do, cannot do, and should do next. In: Proc. of the 4th Workshop on NLP for Conversational AI. Dublin: ACL, 2022. 148–165. [doi: 10.18653/v1/2022.nlp4convai-1.13]

[85] Feng Y, Shi QK, Gao XY, Wan J, Fang CR, Chen ZY. DeepGini: Prioritizing massive tests to enhance the robustness of deep neural networks. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 177–188. [doi: 10.1145/3395363.3397357]

[86] Papineni K, Roukos S, Ward T, Zhu WJ. BLEU: A method for automatic evaluation of machine translation. In: Proc. of the 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia: ACL, 2002. 311–318. [doi: 10.3115/1073083.1073135]

[87] Banerjee S, Lavie A. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In: Proc. of the 2005 ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization. Ann Arbor: ACL, 2005. 65–72.

[88] Przybocki M, Peterson K, Bronsart S, Sanders G. The NIST 2008 metrics for machine translation challenge—Overview, methodology, metrics, and results. Machine Translation, 2009, 23(2): 71–103. [doi: 10.1007/s10590-009-9065-6]

[89] Asyrofi MH, Yang Z, Yusuf INB, Kang HJ, Thung F, Lo D. BiasFinder: Metamorphic test generation to uncover bias for sentiment analysis systems. IEEE Trans. on Software Engineering, 2022, 48(12): 5087–5101. [doi: 10.1109/TSE.2021.3136169]

[90] Yagcioglu S, Erdem A, Erdem E, Ikizler-Cinbis N. RecipeQA: A challenge dataset for multimodal comprehension of cooking recipes. In: Proc. of the 2018 Conf. on Empirical Methods in Natural Language Processing. Brussels: ACL, 2018. 1358–1368. [doi: 10.18653/v1/D18-1166]

[91] Labied M, Belangour A, Banane M, Erraissi A. An overview of automatic speech recognition preprocessing techniques. In: Proc. of the 2022 Int'l Conf. on Decision Aid Sciences and Applications (DASA). Chiangrai: IEEE, 2022. 804–809. [doi: 10.1109/DASA54658.2022.9765043]

[92] Asyrofi MH, Thung F, Lo D, Jiang LX. CrossASR: Efficient differential testing of automatic speech recognition via text-to-speech. In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Adelaide: IEEE, 2020. 640–650. [doi: 10.1109/ICSME46990.2020.00066]

[93] Asyrofi MH, Yang Z, Lo D. CrossASR++: A modular differential testing framework for automatic speech recognition. In: Proc. of the 29th ACM Joint Meeting European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM,

2021. 1575–1579. [doi: 10.1145/3468264.3473124]

[94] Ji P, Feng Y, Liu J, Zhao ZH, Chen ZY. ASRTest: Automated testing for deep-neural-network-driven speech recognition systems. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 189–201. [doi: 10.1145/3533767.3534391]

[95] Yuen DHX, Pang AYC, Yang Z, Chong CY, Lim MK, Lo D. ASDF: A differential testing framework for automatic speech recognition systems. arXiv:2302.05582, 2023.

[96] Rajan SS, Udeshi S, Chattopadhyay S. AequeVox: Automated fairness testing of speech recognition systems. In: Proc. of the 25th Int'l Conf. on Fundamental Approaches to Software Engineering. Munich: Springer, 2022. 245–267. [doi: 10.1007/978-3-030-99429-7_14]

[97] Errattahi R, El Hannani A, Ouahmane H. Automatic speech recognition errors detection and correction: A review. Procedia Computer Science, 2018, 128: 32–37. [doi: 10.1016/j.procs.2018.03.005]

[98] Chen XZ, Ma HM, Wan J, Li B, Xia T. Multi-view 3D object detection network for autonomous driving. In: Proc. of the 2017 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). Honolulu: IEEE, 2017. 6526–6534. [doi: 10.1109/CVPR.2017.691]

[99] Guo YL, Wang HY, Hu QY, Liu H, Liu L, Bennamoun M. Deep learning for 3D point clouds: A survey. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2021, 43(12): 4338–4364. [doi: 10.1109/TPAMI.2020.3005434]

[100] Yang B, Luo WJ, Urtasun R. PIXOR: Real-time 3D object detection from point clouds. In: Proc. of the 2018 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Salt Lake City: IEEE, 2018. 7652–7660. [doi: 10.1109/CVPR.2018.00798]

[101] Levinson J, Askeland J, Becker J, Dolson J, Held D, Kammel S, Kolter JZ, Langer D, Pink O, Pratt V, Sokolsky M, Stanek G, Stavens D, Teichman A, Werling M, Thrun S. Towards fully autonomous driving: Systems and algorithms. In: Proc. of the 2011 IEEE Intelligent Vehicles Symp. (IV). Baden-Baden: IEEE, 2011. 163–168. [doi: 10.1109/IVS.2011.5940562]

[102] Wang ZR, Yang CG, Ju ZJ, Li ZJ, Su CY. Preprocessing and transmission for 3D point cloud data. In: Proc. of the 10th Int'l Conf. on Intelligent Robotics and Applications. Wuhan: Springer, 2017. 438–449. [doi: 10.1007/978-3-319-65289-4_42]

[103] Yue XY, Wu BC, Seshia SA, Keutzer K, Sangiovanni-Vincentelli AL. A LiDAR point cloud generator: From a virtual world to autonomous driving. In: Proc. of the 2018 ACM Int'l Conf. on Multimedia Retrieval. Yokohama: ACM, 2018. 458–464. [doi: 10.1145/3206025.3206080]

[104] Guo A, Feng Y, Chen ZY. LiRTest: Augmenting LiDAR point clouds for automated testing of autonomous driving systems. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 480–492. [doi: 10.1145/3533767.3534397]

[105] Chao QW, Bi HK, Li WZ, Mao TL, Wang ZQ, Lin MC, Deng ZG. A survey on visual traffic simulation: Models, evaluations, and applications in autonomous driving. Computer Graphics Forum, 2020, 39(1): 287–308. [doi: 10.1111/cgf.13803]

[106] O'Kelly M, Sinha A, Namkoong H, Duchi J, Tedrake R. Scalable end-to-end autonomous vehicle testing via rare-event simulation. In: Proc. of the 32nd Int'l Conf. on Neural Information Processing Systems. Montréal: Curran Associates Inc., 2018. 9849–9860.

[107] Medrano-Berumen C, Akbaş MI. Abstract simulation scenario generation for autonomous vehicle verification. In: Proc. of the 2019 SoutheastCon. Huntsville: IEEE, 2019. 1–6. [doi: 10.1109/SoutheastCon42311.2019.9020575]

[108] Chao QW, Jin XG, Huang HW, Foong S, Yu LF, Yeung SK. Force-based heterogeneous traffic simulation for autonomous vehicle testing. In: Proc. of the 2019 Int'l Conf. on Robotics and Automation (ICRA). Montreal: IEEE, 2019. 8298–8304. [doi: 10.1109/ICRA.2019.8794430]

[109] Kim B, Masuda T, Shiraishi S. Test specification and generation for connected and autonomous vehicle in virtual environments. ACM Trans. on Cyber-physical Systems, 2019, 4(1): 8. [doi: 10.1145/3311954]

[110] Han JC, Zhou ZQ. Metamorphic fuzz testing of autonomous vehicles. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering Workshops. Seoul: ACM, 2020. 380–385. [doi: 10.1145/3387940.3392252]

[111] Fremont DJ, Kim E, Pant YV, Seshia SA, Acharya A, Bruso X, Wells P, Lemke S, Lu Q, Mehta S. Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In: Proc. of the 23rd IEEE Int'l Conf. on Intelligent Transportation Systems. Rhodes: IEEE, 2020. 1–8. [doi: 10.1109/ITSC45102.2020.9294368]

[112] Majumdar R, Mathur A, Pirron M, Stegner L, Zufferey D. PARACOSM: A test framework for autonomous driving simulations. In: Proc. of the 24th Int'l Conf. Fundamental Approaches to Software Engineering. Luxembourg City: Springer, 2021. 172–195. [doi: 10.1007/978-3-030-71500-7_9]

[113] Nguyen V, Huber S, Gambi A. SALVO: Automated generation of diversified tests for self-driving cars from existing maps. In: Proc. of the 2021 IEEE Int'l Conf. on Artificial Intelligence Testing. Oxford: IEEE, 2021. 128–135. [doi: 10.1109/AITEST52744.2021.00033]

[114] Li CW, Cheng CH, Sun TT, Chen YH, Yan RJ. ComOpT: Combination and optimization for testing autonomous driving systems. In: Proc. of the 2022 Int'l Conf. on Robotics and Automation. Philadelphia: IEEE, 2022. 7738–7744. [doi: 10.1109/ICRA46639.2022.9811794]

[115] Ding YR, Zou JH, Fan YX, Wang SJ, Liao QM. A digital twin-based testing and data collection system for autonomous driving in extreme traffic scenarios. In: Proc. of the 6th Int'l Conf. on Video and Image Processing. Shanghai: ACM, 2023. 101–109. [doi: 10.1145/3579109.3579127]

[116] Song QY, Runeson P, Persson S. A scenario distribution model for effective and efficient testing of autonomous driving systems. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2023. 215. [doi: 10.1145/3551349.3563239]

[117] Zhou R, Liu YP, Zhang K, Yang O. Genetic algorithm-based challenging scenarios generation for autonomous vehicle testing. IEEE Journal of Radio Frequency Identification, 2022, 6: 928–933. [doi: 10.1109/JRFID.2022.3223092]

[118] Sun Y, Poskitt CM, Sun J, Chen YQ, Yang ZJ. LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2023. 62. [doi: 10.1145/3551349.3556897]

[119] Zhang XD, Zhao W, Sun Y, Sun J, Shen YL, Dong XW, Yang ZJ. Testing automated driving systems by breaking many laws efficiently. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 942–953. [doi: 10.1145/3597926.3598108]

[120] Huai YQ, Chen YTY, Almanee S, Ngo T, Liao X, Wan ZW, Chen QA, Garcia J. Doppelgänger test generation for revealing bugs in autonomous driving software. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Melbourne: IEEE, 2023. 2591–2603. [doi: 10.1109/ICSE48619.2023.00216]

[121] Zhou Y, Lin GJ, Tang Y, Yang KR, Jing W, Zhang P, Chen JB, Gong L, Liu Y. FLYOVER: A model-driven method to generate diverse highway interchanges for autonomous vehicle testing. In: Proc. of the 2023 IEEE Int'l Conf. on Robotics and Automation (ICRA). London: IEEE, 2023. 11389–11395. [doi: 10.1109/ICRA48891.2023.10160868]

[122] Zhang XD, Cai Y. Building critical testing scenarios for autonomous driving from real accidents. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 462–474. [doi: 10.1145/3597926.3598070]

[123] Wang S, Sheng ZH, Xu JW, Chen TL, Zhu JJ, Zhang SH, Yao Y, Ma XX. ADEPT: A testing platform for simulated autonomous driving. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2023. 150. [doi: 10.1145/3551349.3559528]

[124] Tian HX, Wu GQ, Yan JR, Jiang Y, Wei J, Chen W, Li S, Ye D. Generating critical test scenarios for autonomous driving systems via influential behavior patterns. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2023. 46. [doi: 10.1145/3551349.3560430]

[125] Yang YH, Kujanpää K, Babadi IA, Pajarinen J, Ilin A. Suicidal pedestrian: Generation of safety-critical scenarios for autonomous vehicles. In: Proc. of the 26th IEEE Int'l Conf. on Intelligent Transportation Systems (ITSC). Bilbao: IEEE, 2023. 1983–1988. [doi: 10.1109/ITSC57777.2023.10422034]

[126] Hou-Liu J, Jiang ZK, Babikian AA. Concretize: A model-driven tool for scenario-based autonomous vehicle testing. In: Proc. of the 27th ACM/IEEE Int'l Conf. on Model Driven Engineering Languages and Systems. Linz: ACM, 2024. 66–70. [doi: 10.1145/3652620.3687793]

[127] Tang SC, Zhang ZY, Zhou JX, Lei L, Zhou Y, Xue YX. LeGEND: A top-down approach to scenario generation of autonomous driving systems assisted by large language models. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 1497–1508. [doi: 10.1145/3691620.3695520]

[128] Guo A, Zhou Y, Tian HX, Fang CR, Sun YJ, Sun WS, Gao XY, Luu AT, Liu Y, Chen ZY. SoVAR: Build generalizable scenarios from accident reports for autonomous driving testing. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 268–280. [doi: 10.1145/3691620.3695037]

[129] Gao XY, Wang ZJ, Feng Y, Ma L, Chen ZY, Xu BW. MultiTest: Physical-aware object insertion for testing multi-sensor fusion perception systems. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 139. [doi: 10.1145/3597503.3639191]

[130] Pei KX, Cao YZ, Yang JF, Jana S. DeepXplore: Automated whitebox testing of deep learning systems. In: Proc. of the 26th Symp. on Operating Systems Principles. Shanghai: ACM, 2017. 1–18. [doi: 10.1145/3132747.3132785]

[131] Ma L, Juefei-Xu F, Zhang FY, Sun JY, Xue MH, Li B, Chen CY, Su T, Li L, Liu Y, Zhao JJ, Wang YD. DeepGauge: Multi-granularity testing criteria for deep learning systems. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: ACM, 2018. 120–131. [doi: 10.1145/3238147.3238202]

[132] Du XN, Xie XF, Li Y, Ma L, Zhao JJ, Liu Y. DeepCruiser: Automated guided testing for stateful deep learning systems. arXiv:1812.05339, 2018.

[133] Ma L, Juefei-Xu F, Xue MH, Li B, Li L, Liu Y, Zhao JJ. DeepCT: Tomographic combinatorial testing for deep learning systems. In: Proc. of the 26th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Hangzhou: IEEE, 2019. 614–618. [doi: 10.1109/SANER.2019.8668044]

[134] Sun YC, Huang XW, Kroening D, Sharp J, Hill M, Ashmore R. Structural test coverage criteria for deep neural networks. ACM Trans. on Embedded Computing Systems, 2019, 18(5s): 94. [doi: 10.1145/3358233]

[135] Kim J, Feldt R, Yoo S. Guiding deep learning system testing using surprise adequacy. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 1039–1049. [doi: 10.1109/ICSE.2019.00108]

[136] Kim J, Feldt R, Yoo S. Evaluating surprise adequacy for deep learning system testing. ACM Trans. on Software Engineering and Methodology, 2023, 32(2): 42. [doi: 10.1145/3546947]

[137] Gerasimou S, Eniser HF, Sen A, Cakan A. Importance-driven deep learning system testing. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: ACM, 2020. 702–713. [doi: 10.1145/3377811.3380391]

[138] Ma W, Papadakis M, Tsakmalis A, Cordy M, Le Traon Y. Test selection for deep learning systems. ACM Trans. on Software Engineering and Methodology, 2021, 30(2): 13. [doi: 10.1145/3417330]

[139] Xie XY, Yin PB, Chen SQ. Boosting the revealing of detected violations in deep learning testing: A diversity-guided method. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2023. 17. [doi: 10.1145/3551349.3556919]

[140] Sun YC, Huang XW, Kroening D, Sharp J, Hill M, Ashmore R. Testing deep neural networks. arXiv:1803.04792, 2019.

[141] Yoo S, Harman M. Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification and Reliability, 2012, 22(2): 67–120. [doi: 10.1002/stv.430]

[142] Arrieta A. Multi-objective metamorphic follow-up test case selection for deep learning systems. In: Proc. of the 2022 Genetic and Evolutionary Computation Conf. Boston: ACM, 2022. 1327–1335. [doi: 10.1145/3512290.3528697]

[143] The MNIST dataset. 2023. https://www.tensorflow.org/datasets/catalog/mnist

[144] The CIFAR-10 dataset. 2023. https://www.cs.toronto.edu/~kriz/cifar.html

[145] The ImageNet dataset. 2023. https://www.image-net.org/update-mar-11-2021.php

[146] Xie XF, Ma L, Juefei-Xu F, Xue MH, Chen HX, Liu Y, Zhao JJ, Li B, Yin JX, See S. DeepHunter: A coverage-guided fuzz testing framework for deep neural networks. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Beijing: ACM, 2019. 146–157. [doi: 10.1145/3293882.3330579]

[147] Sun YC, Wu M, Ruan WJ, Huang XW, Kwiatkowska M, Kroening D. Concolic testing for deep neural networks. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: ACM, 2018. 109–119. [doi: 10.1145/3238147.3238172]

[148] Riccio V, Tonella P. Model-based exploration of the frontier of behaviours for deep learning system testing. In: Proc. of the 28th ACM Joint Meeting European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 876–888. [doi: 10.1145/3368089.3409730]

[149] Zohdinasab T, Riccio V, Tonella P. DeepAtash: Focused test generation for deep learning systems. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 954–966. [doi: 10.1145/3597926.3598109]

[150] Hu Q, Guo YJ, Xie XF, Cordy M, Papadakis M, Ma L, Le Traon Y. Aries: Efficient testing of deep neural networks via labeling-free accuracy estimation. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Melbourne: IEEE, 2023. 1776–1787. [doi: 10.1109/ICSE48619.2023.00152]

[151] Chen JJ, Wu Z, Wang Z, You HM, Zhang LM, Yan M. Practical accuracy estimation for efficient deep neural network testing. ACM Trans. on Software Engineering and Methodology, 2020, 29(4): 30. [doi: 10.1145/3394112]

[152] Li Z, Ma XX, Xu C, Cao C, Xu JW, Lü J. Boosting operational DNN testing efficiency through conditioning. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 499–509. [doi: 10.1145/3338906.3338930]

附中文参考文献

[10] 泰国威胁将对 Facebook 采取法律行动 因自动翻译对国王大不敬? 2020. https://k.sina.cn/article_1949671172_74359f0400100rsmq.html

[20] 王赞, 闫明, 刘爽, 陈俊洁, 张栋迪, 吴卓, 陈翔. 深度神经网络测试研究综述. 软件学报, 2020, 31(5): 1255–1275. http://www.jos.org.cn/1000-9825/5951.htm [doi: 10.13328/j.cnki.jos.005951]

[28] 叶仕俊, 张鹏程, 吉顺慧, 戴启印, 袁天昊, 任彬. 人工智能软件系统的非功能属性及其质量保障方法综述. 软件学报, 2023, 34(1): 103–129. http://www.jos.org.cn/1000-9825/6409.htm [doi: 10.13328/j.cnki.jos.006409]

[37] 特斯拉 Autopilot 再致车祸: 不减速直接撞上白色货车. 2020. https://nev.ofweek.com/2020-06/ART-71005-8440-30442524.html

[58] 袁珑, 李秀梅, 潘振雄, 孙军梅, 肖蕾. 面向目标检测的对抗样本综述. 中国图象图形学报, 2022, 27(10): 2873–2896. [doi: 10.11834/jig.210209] [doi: 10.11834/jig.210209]

作者简介

吉品, 博士生, 主要研究领域为智能软件系统测试, 深度学习框架测试.

冯洋, 博士, 副教授, CCF 高级会员, 主要研究领域为软件质量保障.

吴朵, 硕士生, 主要研究领域为深度学习框架测试, 智能软件系统测试.

刘嘉, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为人工智能的测试与优化, 群体智能, 大数据质量.

赵志宏, 博士, 教授, 博士生导师, 主要研究领域为软件工程, 信息系统工程, 机器学习.