arXiv:2303.01302v1 [cs.SE] 2 Mar 2023

基于推理的软件测试

Luca Giamattei、Roberto Pietrantuono、Stefano Russo

DIETI，`那不勒斯费德里科二世大学克劳迪奥街21号，
80125 - 那不勒斯，意大利{luca.giamattei,
roberto.pietrantuono, stefano.russo}@unina.it

摘要——**随着软件系统日益普及且自主化，测试其质量的能力正面临严峻挑战。许多系统需要在不确定且变化迅速的环境中运行，并且常常必须自主做出智能决策。这很容易导致测试时需要探索的状态空间变得难以处理。当前最先进的技术试图跟上这一发展步伐，例如通过某种形式的（显式或隐式）从观察中学习来增强测试人员的直觉，从而高效地搜索这一空间。例如，这些技术会利用历史数据来驱动搜索（如机器学习驱动的测试），或利用测试执行数据本身（如自适应测试或基于搜索的测试）。尽管取得了毋庸置疑的进展，如何在如此庞大的空间中实现更智能的搜索，仍是一个迫切的问题。**

我们提出基于推理的软件测试（RBST），将测试问题视为因果推理任务的一种全新思路。与单纯基于直觉或当前最先进的基于学习的策略相比，我们认为，因果推理能够更自然地模拟人类“智能地”搜索空间的过程。RBST旨在借助计算能力，模仿并增强这种能力。这一概念上的飞跃可以为新一类技术奠定基础：通过利用因果发现与因果推断的大量工具，能够以多种方式实例化所提出的框架。本文报告的初步结果颇具前景。

索引词—**因果推理、软件测试**

I. 引言

软件测试归根结底关乎预测。在推导测试时，测试人员基本上会问自己：什么输入会使系统发生故障？他们会尝试设想，在给定的执行场景中，系统面对某个特定输入时将如何运行。为了获得对这一预测的支持，他们通常依赖一些辅助信息来推导故障暴露测试（例如，使用边界/对抗性输入、覆盖率度量，以及相对于所观察上下文的差异度量/相似度量）。现有技术利用测试人员关于哪些因素预期与故障相关的信念，来实现测试任务的自动化。在基于学习的策略中，这种信念通过以往的观测结果（测试/运行数据）得到佐证和补充，用于训练机器学习（ML）模型 [1]–[3]，或者以无监督方式利用测试执行的反馈，在下一步推导更好的测试（例如，自适应测试[4], 基于搜索的软件测试 [5], [6]）。学习支持多种任务，例如测试生成 [7], [8],测试选择与优先级排序 [9], [10], 测试执行 [11], [12]。

当前这种测试方法存在固有局限：要么仅凭直觉（专业知识/经验）“猜测”正确的与故障相关的信息，再据此设计技术；要么专注于从以往的观察中学习故障模式，预测下一步最合适的测试，但前提是假设未来的情境与过去相似。

原则上，人类推理很擅长“猜测”与故障相关的信息，但显然无法扩展到足够大的规模。从过去学习确实有助于在搜索空间中导航，但这只是一种权宜之计：在特定情境下对观察结果学到的相关性，可用于仅根据已观察到的内容做出“预测” [13]。这掩盖了一个概念上的缺陷：借助当前的基于学习的技术（如机器学习驱动的测试），我们实际上把什么会导致它故障？这个问题转变成了什么与故障的相关性更高？但这并不是测试人员试图做出的那种预测：提出这个问题时，他们真正想问的是什么输入会导致系统故障？

前一个问题可以表述为：在我学习模型时所处的相同情境中（即相同的数据分布下），如果输入 Y 恰好等于 X，那么我们预期会得到什么输出 x？基于~~此，例如，~~我们会选择或优先考虑那些与过去观察到的导致故障的测试更相似的测试，而机器学习非常擅长这项任务。

因果问题可以表述为：如果输入 Y 被主动设为 X，我们预期会得到什么输出 x （也就是说，如果我们改变 X 的分布）？测试人员提出的问题本质上是因果问题，但像机器学习这样基于关联的学习范式无法处理因果性 [14]。它确实增强了我们的模式搜索能力，这是很大的附加价值；然而，我们还能做得更多，进一步增强因果推理能力。后者不局限于从观察中学习，还能通过对感兴趣的变量提出干预来进行学习。

这项工作提出了一种概念上的飞跃，改变机器与人类协作的方式，使双方能够智能地探索庞大的搜索空间并推导出测试。机器应当支持并提升人类的推理能力，而不仅仅是在以往的观察中搜索模式。我们认为，在软件测试中，自动化因果推理是我们应当迈出的下一步，因为它使我们能够主动推断知识，并在不同于已观察世界的情境中提出如果.....会发生什么的问题。

下面，我们提出将测试问题表述为因果推理任务的方法，并将其称为基于推理的软件测试（RBST）。随后，我们对 RBST 的一个基本实例进行初步评估，将其用于测试自动驾驶系统，并与自适应测试和一种由机器学习驱动的搜索技术进行比较。结果表明，利用因果关系推导违反安全性的测试具有优势。

II. 背景

A. 因果推断

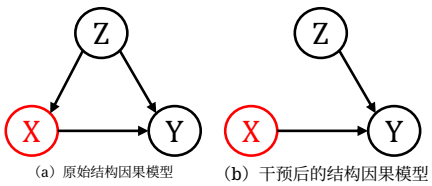
因果性是指一个事件促成其他事件产生的影响 [15]。数十年来，统计学家一直试图通过统计方法解释因果性，即识别变量之间的关联（例如相关性、回归）；然而，这些方法无法区分原因与效应。它们仅限于 Pearl 所称的因果阶梯第一层 [13], ，即“关联”。因果推理使我们能够上升到第二层和第三层，分别通过“行动”（干预）和“想象”（反事实）来学习。因果性研究有两个主要分支：因果推断旨在量化改变一个或多个变量对关注结果的效应；因果发现旨在从观察数据中提取因果模型。因果模型是变量间因果关系的数学表示。一种常见的模型类型定义如下：

定义 1. （结构因果模型）*SCM* 是一个有向无环图 $\mathcal{G} = (\boldsymbol{X}, \mathcal{E})$，其中节点 $\in \boldsymbol{X}$ 是随机变量，边 $\in \mathcal{E}$ 是它们之间的因果关系，并与一组结构赋值$X_k := f_k(Pa(X_k)\ U_k)$相关联；这些结构赋值将（内生）随机变量 X_k定义为其父节点 $Pa(X_k)$和（外生）独立随机噪声变量 U_k的函数。

在因果推断中，我们关注的是将变量 W 设为某个值 w （即进行干预）后结果变量 X_k 的分布，而不是像 ML 中那样仅仅观测到某个事件 $W = w$ （即 $P(X_k|W = w)$）之后的分布。Pearl 引入了*do*算子，这是一种对物理干预的数学表示，写作 $P(X_k|do(W_i = w))$[16]。干预 $do(W = w)$会改变 SCM 图（即修改分布），方法是移除其前驱节点与该变量之间的因果关系（即删除 $Pa(W) \to W$ 箭头）。因此： $P(Y|do(X_i = x)) \neq P(Y|(X_i = x))$。

定义 2。（干预分布）*SCM* 上的概率$P(X_k|do(W = w))$是由该 *SCM* 所蕴含的分布，该 *SCM* 是通过将定义 $X_k := f_k(Pa(X_k)\ U_k)$ 替换为 $X_k := w_k$而得到的。

图 1(a) 展示了一个部分结构因果模型，其中*X* 对*Y*有因果影响，而*Z* 同时影响*X* 和*Y*。将*X* 视为干预



（a）原始结构因果模型 （b）干预后的结构因果模型

tion

图 1: （a）结构因果模型示例；（b）对*X*进行干预的效果

变量 W 后，干预会改变图（因而改变分布），方法是移除与其前驱变量之间的因果关系，如图 1(b) 所示。

通过干预，可以借助*do*-演算 [14]估计因果效应：它能够将*do* 操作表示为一组相关变量的条件分布，这些变量可通过图模式（例如后门、前门、工具变量）加以恰当识别。效应可以用多种指标量化，最常见的是平均处理效应 （ATE）[1]。

因果推断的另一个用途是使用反事实。反事实是以虚拟条件句形式表达的命题，例如“如果 W 当时是 w，那么X_k 就会是 x_k”。这使我们能够回答如果当时发生了...会怎样之类的问题，从而探索另一种假设性的过去。

在软件工程中，因果性仅在少数研究中有所应用，例如用于支持根因分析和诊断[17], 调试 [18], 故障定位 [19], [20], ，以及提升机器学习模型的可解释性 [21], [22]。

B. 因果发现

在因果推断中，因果结构通常是预先假定的。构建因果模型主要有三种方法。第一种是对变量进行干预，并观察干预后的概率分布（即受控实验）。还可以进行软 干预，在不将干预变量设为固定值的情况下影响其分布 [23], [24]。

第二种方法是使用因果发现算-法，从观察数据中提取因果结构，从而避免成本高昂（甚至在技术上不可行）的受控实验。过去几十年来，这类算法不断发展，以便更好地利用“大数据”。因果发现算法旨在从观察数据中推断因果结构，并将统计依赖关系（或独立关系）作为因果关系（或缺乏因果关系）的指示 [25]。因果发现算法可分为基于约束的（例如 PC、FCI、RFCI）、基于评分的（例如 GES、FGES、GFCI）和基于函数因果模型的（例如 LinGAM） [26]。

基于约束的算法利用观测数据进行独立性检验，以确定一组边约束[27]。尽管这类算法的优点是适用范围广，但在样本量不大时，表现可能不佳 [26]。基于评分的算法使用贝叶斯信息准则等调整准则，对候选图进行评分并使其得分最大化。拟合优度检验取代了条件独立性检验。最后，基于各种模型假设（例如 LinGAM 的线性非高斯参数化），基于函数因果模型的算法通过确定边的因果方向，尝试识别真实的因果结构。

此外，由于因果模型具有可解释性，也可以结合领域专家的知识手动构建或优化因果模型。

[1]With W binary, ATE $= E[X_k|do(W = 1)] - E[X_k|do(W = 0)]$ [15].

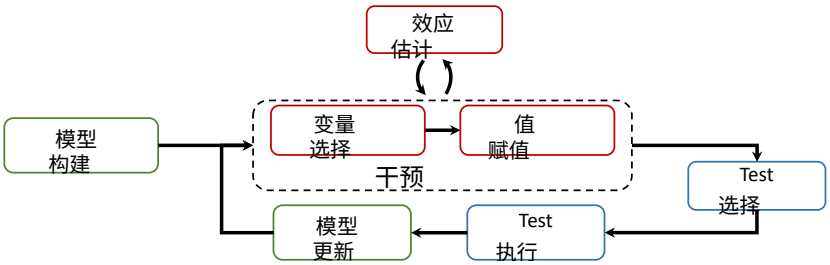


图2：基于推理的软件测试

III. 基于推理的软件测试

因果推断旨在估计一个或多个变量（处理）对某个感兴趣结果的因果效应[15]。在软件测试中，一个直接的应用是测试数据生成，其目标是识别能够使测试输出（例如性能指标或安全相关变量）或某个感兴趣指标（例如覆盖率）最大化或最小化的输入组合。鉴于被测系统的底层因果结构，可以将测试目标重新表述为因果问题。例如，在测试数据生成的背景下，变量间的因果关系使我们能够向模型提出如下问题：“如果我们固定测试输入 $X = x$，测试输出 Y 会发生什么变化？”。如果模型的预测准确，就可以利用模型探索测试空间，而无需执行任何测试，从而筛选并实际运行最有希望的测试。

本文重点关注测试用例生成（并有实验结果支持）。不过，因果推理还可协助测试人员完成许多其他任务，例如回归测试选择与优先级排序、测试套件最小化和测试规划，甚至可借助反事实分析测试执行结果（并支持调试），例如：如果我们修复了 $X = x$，我们是否仍会观察到故障？

A. 测试用例生成

高效的测试用例生成需要能够智能地搜索输入空间，以合理的成本找出满足测试目标的组合（例如，最大限度地提高故障检测率或覆盖率）。这一探索过程被重新定义为一项因果推理任务。图2展示了设想中的基于推理的软件测试流程。

实例化该过程有多种可能。表I列出了 RBST 关键步骤的可能替代方案，但并未穷尽所有选项。对替代方案进行组合选择，即可形成一种测试生成策略。

因果模型编码了支持推理的因果结构知识。它最初构建，并随着更多数据的出现而不断迭代完善。如第 II-B 节所述，可采用的方法包括受控（或软）实验、从数据中进行因果结构发现（CSD）（例如，对一批初始测试进行过去的执行），以及非替代性地使用领域知识。在后一种情况下，值得强调的是，模型的有效性随后会通过数据进行交叉验证；如果数据不支持某个假定的关系，领域专家可以完善模型，以用于下一轮迭代。

可以通过一组干预来查询模型。每种干预都可以通过解析或模拟来估计潜在变化的效应。因此，干预会生成一个假设性 测试用例，即对测试的假设，以及执行该测试时预期产生的效应。此处，RBST 用户需要制定策略，以选择要进行干预的变量及其赋值。一种做法是选择能够最大限度地增加信息（即最大限度地减少不确定性）、从而更好地了解真实图的干预——也就是说，通过干预来更好地学习该图 [28]。

另一方面，测试人员可能希望设置能够最大化/最小化所需测试目标（例如，提高覆盖率、产生关键输出）或最大化多样性（例如，同时影响更多效应，这有助于多目标测试）的干预。也可以组合采用这些方式，例如，在模型准确率和奖励之间进行权衡：起初，采用不确定性驱动的干预来改进模型可能更合适，之后再逐步转向目标驱动的干预。

无论选择何种准则，变量和取值的选择/赋值都可以通过多种方式实现，例如：概率抽样（如均匀或非均匀随机抽样）、基于搜索或基于学习的技术、自适应策略（即利用先前的选择来指导后续选择），甚至穷举（具体视情境而定；与实际测试执行时间相比，干预计算时间可能微不足道）。

表I：实例化 RBST 流程的备选方案

Step	描述	备选方案
模型构建	如何构建该因果模型	因果发现 领域专家 受控实验
干预变量选择	如何选择变量用于 the 干预的变量	标准： 不确定性、置信度 测试目标 多样性 策略： 采样 基于搜索 自适应 基于学习 穷举
干预值赋值	如何设置干预值针对所选变量	
效应估计	如何估计某项.....的效应干预	解析 基于模拟

每次干预都会产生一种效应，通常用平均处理效应（ATE）来量化，不过也可能关注其他指标（例如：受处理者平均处理效应（ATT）、条件平均处理效应（CATE））[15]。效应估计通常通过解析方法获得，例如使用 DoWhy 等库[29]，也可以通过模拟获得，即从干预后分布中抽样并计算所需的效应估计。

从如此获得的假设性测试集中选取一个（或多个）实际测试。基本选择是获取效应最大的测试，但也值得探索其他方案，例如提高多样性。随后执行这些测试；这也会丰富知识，以更新模型。最后，可以利用更新后的知识，通过反驳检验和置信区间计算 [29]来再次检查估计效应是否显著（即是否由偶然因素导致）。

IV. 评估

A. 上下文

我们在关键场景生成的自动驾驶系统（ADS）测试背景下，评估 RBST 的一个基本实例。我们使用 Pylot [30] 作为 ADS，并使用 CARLA 作为模拟器 [31]。

测试场景[2] 由以下内容定义：

- 十六个（类别型）变量，其中包括道路类型、是否存在车辆、天气状况等；
- 一个输出，即与其他车辆的最小距离 d 。

测试会话的目标是发现安全违规，即事件 $d = 0$ （碰撞）至少发生一次的场景。

我们在部署于 Google Cloud Compute Engine 平台上的虚拟机中运行实验，[3] 该虚拟机配置了 Ubuntu 18.04 版本。用于因果发现和推断的库分别为 pycausal[4]（基于 TETRAD [32]）和 Dowhy-gcm [33]。

B. 生成策略

我们为 RBST 采用各步骤替代策略（见第 III-A 节）中最保守的选项：

- **模型构建**：我们选择 CSD，并采用 FCI算法[27], ，这是 CSD 最简单的解决方案之一。初始数据库包含 100 个随机生成的测试；
- **干预变量选择**：我们随机选择要进行干预的变量；
- **干预值分配**：我们穷举所选（类别型）变量的所有可能值，并选择使测试目标最小化的值；
- **效应估计**：我们采用基于模拟的方法，为所选变量的每个可能值生成一个假设测试场景。[5]

[2]详情见复现包，地址为：https://github.com/uDEVOPS2020/Replication-package-Reasoning-Based-Software-Testing。[3] https://cloud.google.com/compute。[4]https://zenodo.org/record/3592985。[5] 样本量为 = 1,000，即 dowhy-GCM 的默认值。

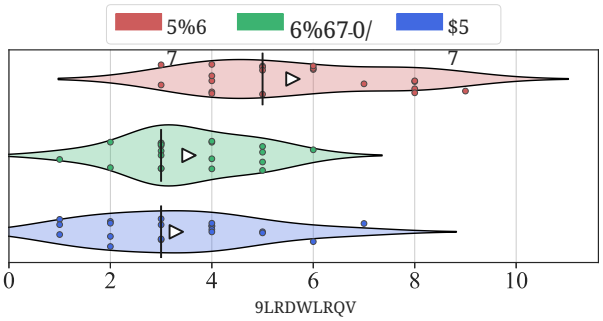


图 3：RBST 与 SBST-ML 和 ART 的效果对比：测试重复期间安全违规数量的小提琴图

每次迭代都会根据已执行的测试更新初始模型。我们将 RBST 与以下方法进行比较：

i) 一种机器学习驱动的基于搜索的技术（SBST-ML），与 [34], 类似，使用代理机器学习模型来节省测试执行时间，并结合遗传算法来演化测试群体（与RBST使用相同的初始数据库）；ii) 自适应随机测试（ART） [35]。

C. 初步发现

对于每种技术，我们都在固定的120分钟时间预算内运行测试会话，并重复20次。评估比较了各技术的有效性（整个测试会话中发现的违规数量）和效率（每 20 分钟时间间隔内发现的违规数量）。此外，我们还研究了各技术在多大程度上将输出值推向“边缘”（即接近阈值），以及所生成测试套件的多样性。

图3展示了三种对比技术（RBST、SBST-ML、ART）在20次重复中发现的违规数量的小提琴图。对于每种技术，黑色竖线表示中位数，小三角形表示平均值。RBST发现的违规数量显著多于其他对比方法。（统计检验结果为：Friedman检验p值 1.37E-03；成对Dunn检验p值：RBST与 SBST-ML相比为1.43E-02；RBST与 ART相比为 2.20E-03。）

图 4 展示了三种技术随时间发现的违规平均数量。由于执行一个场景

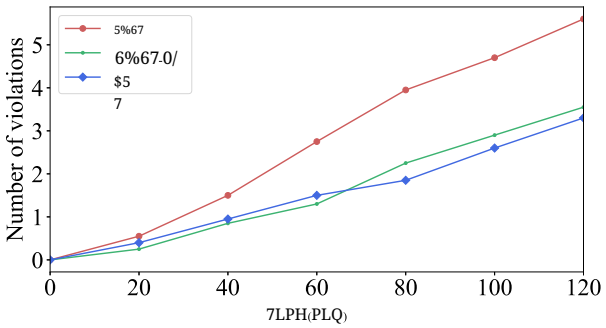


图 4：RBST 与 SBST-ML 和 ART 的效率对比：测试时间内的平均违规次数

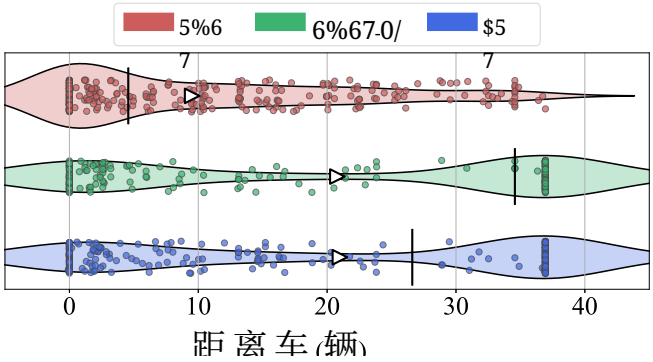


图 5：与其他车辆的距离的小提琴图（接近零的值对应于接近违规的场景）

平均耗时不到 10 分钟，我们将时间预算划分为 20 分钟的区间，以确保每个时间段至少包含 2 个场景。可以看出，RBST 的表现显著优于其他技术，仅用一半的预算就几乎达到了基于搜索技术的最大值。

为研究接近违规的场景，我们收集了每个场景执行的输出值。由于该输出对应于与其他车辆的最小距离，因此接近零的值表示接近违反安全要求的测试场景。图 5 展示了三种对比技术的结果小提琴图。RBST 的表现最佳（Friedman p 值 3.23E-13；Dunn p 值s： <1.00E-04 与 SBST-ML 和 ART 均相比），而 SBST-ML 与 ART 之间没有差异（Dunn$p$ 值 = 1.00E+00）。

图 6 报告了测试套件的多样性，该多样性通过输入上的测试集直径（TSD-I）进行估算；TSD-I 是一种广为人知的黑盒测试套件多样性度量指标 [36]–[38]。RBST 的中位数和均值表现最差，但差异并不显著（Friedman$p$ 值：2.93E-01）。

由于因果推断步骤，RBST 会带来时间开销。平均而言，生成一个测试用例需要 t_{gen}=1.44 秒；相较于平均测试执行时间（t_{gen} = 0.34% t_{exec}），这一开销可以忽略不计。

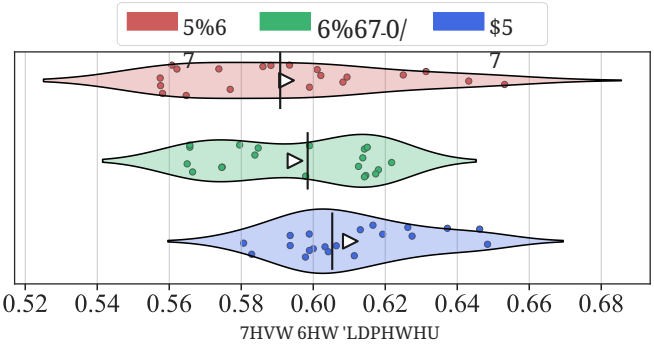


图 6：测试套件多样性方面的技术比较：输入测试重复次数下测试集直径的小提琴图

V. 未来计划

基于推理的软件测试旨在促进因果推理在软件测试中的应用。根据测试任务（例如测试生成、测试优先级排序）、目标（例如故障检测、覆盖率），尤其是 RBST 流程步骤的具体实现方式（表I），未来几年可以实现多种技术。

我们的短期计划是探索用于测试生成的 RBST 替代实例化方式，从而为其最佳实现提供判据。例如，通过在增强模型（如不确定性最小化）与实现测试目标之间选择适当的权衡，可以显著提升性能。

探索可能干预空间的策略也是一个很大的改进机会。随后，我们将研究因果模型的可解释性，因为这有助于无缝整合人类知识。

从中长期来看，我们计划针对回归测试，并探索多个测试目标。此外，我们旨在将反事实推理纳入测试执行后的分析流程，以便通过 实际因果分析支持调试（即为已发生事件分配因果责任）。最后，我们计划将 RBST 的应用拓展到各种领域，以增强外部效度。

致谢

本项目获得了欧盟 Horizon 2020 研究与创新计划的资助，该计划依据玛丽·斯克沃多夫斯卡-居里资助协议第871342号实施。本项目还得到了 DIETI COSMIC项目的支持。

参考文献

[1] R. Ben Abdessalem、S. Nejati、L. C. Briand 和 T. Stifter，《使用可学习进化算法测试基于视觉的控制系统》，载于第*40*届软件工程国际会议（*ICSE*）。ACM，2018年，第1016–1026页。[2] V. H. S. Durelli、R. S. Durelli、S. S. Borges、A. T. Endo、M. M. Eler、D. R. C. Dias 和 M. P. Guimar˜aes，《应用于软件测试的机器学习：一项系统映射研究》，*IEEE*可靠性汇刊，第68卷，第3期，第1189–1212页，2019年9月。[3] Y. Pang、X. Xue 和 A. S. Namin，《通过 k均值聚类识别有效测试用例以增强回归测试》，载于第*12*届机器学习与应用国际会议（*ICMLA*），第2卷。IEEE，2013年，第78–83页。[4] G. E. Mullins、P. G. Stankiewicz、R. C. Hawthorne 和 S. K. Gupta，《为自动驾驶车辆的测试与评估自适应生成高难度场景》，《系统与软件杂志》，第137卷，第197–215页，2018年。[5] P. McMinn，《基于搜索的软件测试：过去、现在与未来》，载于*IEEE*第四届软件测试、验证与确认国际会议研讨会。IEEE，2011年，第153–163页。[6] M. Harman、Y. Jia 和 Y. Zhang，《基于搜索的软件测试：成果、未决问题与挑战》，载于*2015*年 *IEEE* 第*8*届软件测试、验证与确认国际会议（*ICST*）。IEEE，2015年，第1–12页。[7] J. ˇCegiˇn，《面向安全关键软件的机器学习测试数据生成》，载于第*28*届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议（*ESEC/FSE*）。ACM，2020年，第1678–1681页。[8] Y. Li、Z. Yang、Y. Guo 和 X. Chen，《Humanoid：一种用于自动化黑盒安卓应用测试的深度学习方法》，载于*2019*年第*34*届 *IEEE/ACM* 自动化软件工程国际会议（*ASE*）。IEEE，2019年，第1070–1073页。

[9] G. Grano、T. V. Titov、S. Panichella 和 H. C. Gall，《它会有多高？使用机器学习模型预测自动化测试中的分支覆盖率》，载于*IEEE* 软件质量评估机器学习技术研讨会（*MaLTeSQuE*）。IEEE，2018年，第19–24页。[10] B. Busjaeger 和 T. Xie，《面向测试优先级排序的学习：一项工业案例研究》，载于第*24*届 *ACM SIGSOFT* 软件工程基础国际研讨会（*FSE*）。ACM，2016年，第975–980页。[11] A. Stocco、M. Weiss、M. Calzana 和 P. Tonella，《自动驾驶系统的异常行为预测》，载于第 *42* 届软件工程国际会议（*ICSE*）。ACM，2020年，第359–371页。[12] A. Stocco 和 P. Tonella，《置信度驱动的加权再训练：预测自动驾驶系统中的安全关键故障》《软件：演化与过程杂志》，第34卷，第10期，文章 e2386，2022年。[13] J. Pearl 和 D. Mackenzie，《为什么之书：因果关系的新科学》，第1版。美国：Basic Books, Inc.，2018年。[14] J. Pearl，《因果关系：模型、推理与推断》，第2版。美国：剑桥大学出版社，2009年。[15] A. R. Nogueira、A. Pugnana、S. Ruggieri、D. Pedreschi 和 J. Gama，《因果发现与因果推断的方法和工具》，*WIREs* 数据挖掘与知识发现，第12卷，第2期，文章 e1449，2022年。[16] J. Pearl，《统计学中的因果推断：概述》，《统计调查》，第3卷，第96–146页，2009年。[17] L. Wu、J. Tordsson、E. Elmroth 和 O. Kao，《用于微服务性能诊断的因果推断技术：评估与指导性建议》，载于*IEEE* 自主计算与自组织系统国际会议（*ACSOS*）。IEEE，2021年，第21–30页。[18] B. Johnson、Y. Brun 和 A. Meliou，《因果测试：理解缺陷的根本原因》，载于第*42*届软件工程国际会议（*ICSE*）。ACM，2020年，第87–99页。[19] G. K. Baah、A. Podgurski 和 M. Harrold，《消除程序依赖关系的混杂效应，以实现有效的故障定位》，载于第*19*届 *ACM SIGSOFT* 研讨会暨第*13*届欧洲软件工程基础会议（*ESEC/FSE*）。ACM，2011年，第146–156页。[20] Y. K¨uç¨uk、T. A. D. Henderson 和 A. Podgurski，《通过整合基于值和谓词的因果推断技术改进故障定位》，载于第*43*届软件工程国际会议（*ICSE*）。IEEE，2021年，第649–660页。[21] B. Sun、J. Sun、L. H. Pham 和 J. Shi，《基于因果关系的神经网络修复》，载于第*44*届软件工程国际会议（*ICSE*）。ACM，2022年，第338–349页。[22] A. Chattopadhyay、P. Manupriya、A. Sarkar 和 V. N. Balasubramanian，《神经网络归因：一种因果视角》，载于第*36*届机器学习国际会议，《机器学习研究论文集》系列，第97卷。PMLR，2019年，第981–990页。[23] F. Eberhardt 和 R. Scheines，《干预与因果推断》，《科学哲学》，第74卷，第5期，第981–995页，2007年。[24] M. Kocaoglu 、 A. Jaber 、 K. Shanmugam 和 E. Bareinboim，《通过软干预表征并学习含潜变量的因果图》，载于第*32*届神经信息处理系统大会（*NeurIPS*），H. Wallach、H. Larochelle、A. Beygelzimer、F. d'Alch´e-Buc、E. Fox 和 R. Garnett 编。Curran Associates, Inc.，2019年，第14 346–14 356页。[25] F. Eberhardt，《因果发现基础导论》，《国际数据科学与分析杂志》，第3卷第2期，第81–91页，2017年。[26] C. Glymour、K. Zhang和P. Spirtes，《基于图模型的因果发现方法综述》，《遗传学前沿》，第10卷，2019年06月。[27] P. Spirtes、C. Glymour和R. Scheines，《因果、预测与搜索》，第2版，自适应计算与机器学习丛书。美国马萨诸塞州剑桥：麻省理工学院出版社，2001年。[28] M. Steyvers、J. B. Tenenbaum、E. Wagenmakers和B. Blum，《从观察和干预中推断因果网络》，《认知科学》，第27卷第3期，第453–489页，2003年。[29] A. Sharma、C. Zhang、V. Syrgkanis和E. Kiciman，《DoWhy：应对因果假设表达与验证中的挑战》，ICML 2021年因果推断中被忽视的假设研讨会，技术报告MSR-TR-2021-15，2021年7月。[30] I. Gog、S. Kalra、P. Schafhalter、M. A. Wright、J. E. Gonzalez和I. Stoica，《Pylot：用于探索自动驾驶车辆延迟-准确率权衡的模块化平台》，载于*IEEE*国际机器人与自动化会议（*ICRA*）。IEEE，2021年，第8806–8813页。[31] A. Dosovitskiy、G. Ros、F. Codevilla、A. Lopez和V. Koltun，《CARLA：开放式城市驾驶模拟器》，载于第*1*届机器人学习年会。PMLR，2017年，第1–16页。[32] J. Ramsey、K. Zhan、M. Glymour、R. S. Romero、B. Huang、I. Ebert-Uphoff、S. M. Samarasinghe、E. A. Barnes和C. Glymour，《TETRAD——因果发现工具箱》，载于第*8*届国际气候信息学研讨会，2018年。[33] P. Bl¨obaum、P. G¨otz、K. Budhathoki、A. A. Mastakouri和D. Janzing，《Dowhy-gcm：用于图形因果模型中因果推断的Dowhy扩展》，2022年。[34] F. U. Haq、D. Shin和L. Briand，《使用代理辅助和多目标优化对DNN系统进行高效在线测试》，载于第*44*届软件工程国际会议（*ICSE*）。ACM，2022年，第811–822页。[35] T. Y. Chen、F.-C. Kuo、R. G. Merkel和T. Tse，《自适应随机测试：测试用例多样性的艺术》，《系统与软件杂志》，第83卷第1期，第60–66页，2010年。[36] R. Feldt、S. Poulding、D. Clark和S. Yoo，《测试集直径：量化测试用例集合的多样性》，载于*IEEE*国际软件测试、验证与确认会议（*ICST*）。IEEE，2016年，第223–233页。[37] B. Miranda、R. Verdecchia、E. Cruciani和A. Bertolino，《可扩展的基于相似度的测试用例优先级排序的FAST方法》，载于第*40*届软件工程国际会议（*ICSE*）。ACM，2018年，第222–232页。[38] C. Henard、M. Papadakis、M. Harman、Y. Jia和Y. Le Traon，《白盒与黑盒测试优先级排序的比较》，载于第*38*届软件工程国际会议（*ICSE*）。ACM，2016年，第523–534页。