软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(10):4645−4659 [doi: 10.13328/j.cnki.jos.007320] [CSTR: 32375.14.jos.007320]

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

干扰惰性序列的连续决策模型模糊测试*

吴泊逾[1], 王凯锐[1,2,3], 王亚文[1,2,3], 王俊杰[1,2,3]

[1](中国科学院大学, 北京 100049)

[2](智能博弈重点实验室 (中国科学院 软件研究所), 北京 100190)

[3](中国科学院 软件研究所, 北京 100190)

通信作者: 王亚文, E-mail: yawen2018@iscas.ac.cn

摘　要: 人工智能技术的应用已经从分类、翻译、问答等相对静态的任务延伸到自动驾驶、机器人控制、博弈等需要和环境进行一系列“交互-行动”才能完成的相对动态的任务. 执行这类任务的模型核心是连续决策算法, 由于面临更高的环境和交互的不确定性, 而且这些任务往往是安全攸关的系统, 其测试技术面临极大的挑战. 现有的智能算法模型测试技术主要集中在单一模型的可靠性、复杂任务多样性测试场景生成、仿真测试等方向, 对连续决策模型的“交互-行动”决策序列没有关注, 导致无法适应, 或者成本效益低下. 提出一个干预惰性“交互-行动”决策序列执行的模糊测试方法 IIFuzzing, 在模糊测试框架中, 通过学习“交互-行动”决策序列模式, 预测不会触发失效事故的惰性“交互-行动”决策序列, 并中止这类序列的测试执行, 以提高测试效能. 在 4 种常见的测试配置中进行实验评估, 结果表明, 与最新的针对连续决策模型的模糊测试相比, IIFuzzing 可以在相同时间内多探测 16.7%–54.5% 的失效事故, 并且事故的多样性也优于基线方法.

关键词: 连续决策模型; 马尔可夫决策过程; 模糊测试

中图法分类号: TP311

中文引用格式: 吴泊逾, 王凯锐, 王亚文, 王俊杰. 干扰惰性序列的连续决策模型模糊测试. 软件学报, 2025, 36(10): 4645–4659. http://www.jos.org.cn/1000-9825/7320.htm

英文引用格式: Wu BY, Wang KR, Wang YW, Wang JJ. Fuzz Testing for Sequential Decision-making Model with Intervening Inert Sequences. Ruan Jian Xue Bao/Journal of Software, 2025, 36(10): 4645–4659 (in Chinese). http://www.jos.org.cn/1000-9825/7320.htm

Fuzz Testing for Sequential Decision-making Model with Intervening Inert Sequences

WU Bo-Yu[1], WANG Kai-Rui[1,2,3], WANG Ya-Wen[1,2,3], WANG Jun-Jie[1,2,3]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)

[2](State Key Laboratory of Intelligent Game (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[3](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: The application of artificial intelligence technology has extended from relatively static tasks such as classification, translation, and question answering to relatively dynamic tasks that require a series of “interaction-action” with the environment to be completed, like autonomous driving, robotic control, and games. The core of the model for executing such tasks is the sequential decision-making (SDM) algorithm. As it faces higher uncertainties of the environment and interaction and these tasks are often safety-critical systems, the testing techniques are confronted with great challenges. The existing testing technologies for intelligent algorithm models mainly focus on the reliability of a single model, the generation of diverse test scenarios for complex tasks, simulation testing, etc., while no attention is paid to the “interaction-action” decision sequence of the SDM model, leading to unadaptability or low cost-effectiveness. In this study, a fuzz testing method named IIFuzzing for intervening in the execution of inert “interaction-action” decision sequences is proposed. In the fuzz testing framework, by learning the “interaction-action” decision sequence pattern, the inert “interaction-action” decision sequences that will

* 基金项目: 国家自然科学基金 (62232016, 62072442); 中国科学院青年创新促进会; 中国科学院软件研究所基础研究项目 (ISCAS-JCZD-202304); 中国科学院软件研究所创新基金重大重点项目 (ISCAS-ZD-202302); 中国科学院软件研究所 2024 年度“创新团队”(2024-66)
收稿时间: 2024-06-04; 修改时间: 2024-08-07, 2024-09-24; 采用时间: 2024-10-28; jos 在线出版时间: 2025-03-26
CNKI 网络首发时间: 2025-03-27

not trigger failure accidents are predicted and the testing execution of such sequences is terminated to improve the testing efficiency. The experimental evaluations are conducted in four common test configurations, and the results show that compared with the latest fuzz testing for SDM models, IIFuzzing can detect 16.7%–54.5% more failure accidents within the same time, and the diversity of accidents is also better than that of the baseline approach.

Key words: sequential decision-making (SDM) model; Markov decision process (MDP); fuzz testing

随着信息化、智能化程度的发展, 智能系统在国防建设、社会生活中扮演着愈发重要的作用, 例如无人机、水下无人艇、无人坦克、机器人控制、自动驾驶、兵棋推演等. 随着应用场景的愈发复杂, 智能系统也由无需和环境交互的、单功能并相对静态的智能系统演变为复杂的、需要不断和环境交互并实时决策的、相对动态的智能系统. 其中不仅有基于独立事件的预测、分类等模型, 还有需要在有限时间内连续针对环境的反馈做出一系列的响应, 最终完成任务目标的决策模型, 亦即连续决策 (sequential decision-making, SDM) 智能模型. 由于连续决策模型需要和环境不断交互并作出响应, 所以连续决策过程通常可以建模为马尔可夫决策过程 (Markov decision process, MDP)[1]. 连续决策智能体 (SDM-Agent) 通过感知环境得到当前时刻的状态, 根据学到的策略执行动作, 然后得到下一时刻的状态和奖励, 如图 1 所示. SDM-Agent 根据策略不断与环境交互, 产生新的奖励, 再根据奖励改进动作策略, 经过反复的迭代训练, SDM-Agent 就会学到完成任务目标所需要的最优策略.

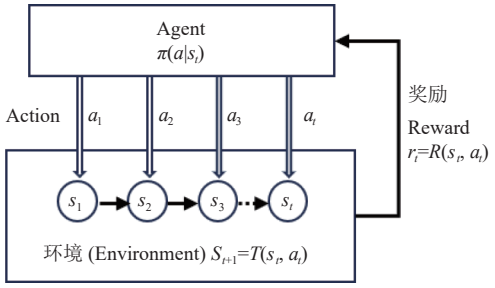


图 1 马尔可夫决策过程模型

尽管这些智能决策模型已经有了很多有效的应用, 但其可靠性、鲁棒性等质量问题依然面临极大的挑战, 我们可以看到, 当前大多数的自动驾驶依然依赖于大量的实际路测, 决策失误引发的事故频发. 据美国加利福尼亚机动车辆管理局的统计[2], 从 2014 年 10 月至 2023 年 4 月, 接到 581 起由于自动驾驶引起的碰撞事故, 导致严重的生命安全和财产损失.

测试一直是保障产品质量的重要手段, 但智能算法的测试由于其影响质量的因素已经不再是算法的功能, 而是算法在模型训练时的泛在性如何去适应应用环境的不确定性, 对人工智能算法的测试非常困难. 近年一些传统的软件测试技术如模糊测试[3]、变异测试[4]、蜕变测试[5,6]等开始尝试改进以用于对人工智能算法的测试, 如图片分类、机器翻译、回答系统等. 这些测试通常是通过对种子测试用例的变异, 产生差异化的测试用例集, 以期望触发算法失效的场景. 对于连续决策算法, 目前常用的测试技术是在人工指导下, 通过手动定义适应度函数 (如自主车辆与行人之间的最小距离等), 然后利用遗传算法生成连续的安全关键场景, 来测试自主智能体的决策能力, 如 AV-FUZZER[7]和 MOSAT[8]等针对自动驾驶系统 (autonomous driving system, ADS) 的测试. 2022 年, Peng 等人提出了一个基于模糊测试的 MDP 测试框架 MDPFuzz[9], 通过选择一组状态作为种子, 然后通过变异产生测试的初始状态, 将每个初始状态输入 MDP 模型, 驱动 SDM-Agent 仿真执行, 直到完成一次仿真或者出现碰撞等事故. MDPFuzz 不需要人工介入, 但存在两个问题: 1) MDPFuzz 将智能算法作为黑盒, 只关注初始状态的选择和变异, 而不考虑决策过程中和环境交互的信息和影响, 导致测试的“交互-行动”决策序列多样性不够充分; 2) 由于测试序列多样性不够, 决策过程非常依赖历史经验而出现惰性, 导致花大量时间执行惯性模式且不能触发失效事故.

我们都知道测试的目的是发现问题, 而并非证明系统是正确的. 对于连续决策模型的测试, 由于环境和交互的不确定性, 发现决策风险远比完成决策任务重要. 本文提出了一种面向 SDM 决策算法, 在模糊测试执行中, 通过干预惰性测试序列, 加速模糊测试触发失效事故的效率和效果的方法 - IIFuzzing (fuzz testing with inert interven-

tion). IIFuzzing 基于通用的模糊测试范式, 首先选择和变异初始状态, 获得测试输入集合, 输入 SDM-Agent 模型, 观测决策过程的状态序列, 分析和预测剩余的执行序列是否会触发失效事故, 干预并中止无失效序列的测试执行. 我们用 3 类最流行并先进的模型 (强化学习、模仿学习、多智能体强化学习), 在自动驾驶、机器人控制以及合作导航等场景进行实验, 实验结果表明 IIFuzzing 在同样的测试时间内, 相比最先进的基线方法 MDPFuzz 可以多触发 17.2%–51.5% 的失效事故, 且事故多样性更好. 我们公开发布了 IIFuzzing 的源代码, 以及在实验中使用的目标模型和模拟环境, 以促进本研究的复现和扩展, 网站链接为 https://github.com/ARMABLE/IIFuzzing. 本文的创新贡献主要包括:

- 提出了一种部分打破连续决策模型黑盒, 通过智能分析和预测, 干预惰性测试序列的模糊测试方法, 可以有效提高测试的效率和测试的多样性.
- 构建了基于状态分布密度的惰性测试序列预测模型, 可以准确地预测惰性测试序列.
- 选择了 4 种模型在 3 种仿真环境进行详实的实验验证, 实验结果表明 IIFuzzing 相比现有的基线方法具有更好的性能.

本文第 1 节介绍深度学习模型测试的相关方法和研究现状. 第 2 节介绍本文所需的基础知识, 包括马尔可夫决策过程和其策略学习模型. 第 3 节介绍本文构建的干扰惰性序列的连续决策模型模糊测试方法. 第 4 节通过对比实验验证了所提方法的有效性. 最后总结全文.

1 相关工作

人工智能技术的发展日新月异, 而与之对应的测试技术却远远没有跟上, 也成为学术界和产业界当前的研究重点, 我们从以下几个方面介绍相关技术的发展.

1.1 面向离散决策模型的测试方法

近年来, 学术界越来越多的研究探索如何将模糊测试、变异测试、蜕变测试等传统软件测试方法应用到对智能模型的测试中. DLFuzz[3]是一个差分模糊测试框架, 通过对测试数据施加微小扰动来变异输入, 以获取最大化的神经元覆盖和预测差异. DeepMutation[4]从源代码和模型两个级别设计了 8 个变异算子, 对深度神经网络的训练数据、训练程序和模型文件进行变异测试. 蜕变测试在测试智能模型方面得到了大规模的应用, 如 TransRepair[10]和 CAT[11]设计了一组蜕变关系来测试机器翻译模型, QAAsker[12]和 QAQA[13]用蜕变测试改进问答系统的能力, Liu 等人基于蜕变测试理论提出了统一的模糊测试框架 QATest[14], 适用于多种形式的问答系统. 从以上工作中可以发现, 传统软件的测试方法经过改进也可以应用于智能模型的测试, 但上述工作针对的被测模型均为离散决策模型, 在针对连续决策的智能模型进行测试时无法生效.

1.2 面向连续决策模型的测试方法

除了上述针对离散决策模型测试的研究, 学术界针对连续决策模型的测试技术也进行了很多探索. 连续决策模型主要应用在自动驾驶、机器人等领域. 目前较为常见的测试方法是生成任务安全攸关的场景, 包括基于知识和基于搜索两种技术路线. 基于知识的方法是基于特定领域的知识本体或蜕变关系生成临界案例, 这些知识来源于人类的驾驶经验, 交通事故报告以及交通法规等, 如 DeepTest[15]和 DeepRoad[6]基于天气变化不应影响自动驾驶系统转向角度的预测等公共知识生成临界测试案例. DeepBillboard[16]通过在真实的广告牌上添加对抗性扰动来生成临界案例, 以检查模型是否输出相同的转向角度. 而基于搜索的方法旨在找到一组测试参数, 在自动驾驶系统中引入诸如碰撞和车道偏离等行为差异. 为了定义参数搜索空间, 大多数基于搜索的方法根据自动驾驶汽车的状态 (例如被测车辆的位置和速度, 以及道路上的其他车辆和行人等) 设计驾驶模式. 搜索过程通常由人工定义的适应度函数 (例如碰撞时间、被测车辆与行人之间的最小距离等) 引导, 利用遗传算法求解这些适应度函数的最优解. 例如, AsFault[17]使用遗传算法来评估道路的长度和位置等参数, 在生成的测试场景中使被测车辆在车道保持任务中更容易出错. FITEST[18]扩展了最近提出的多目标进化算法, 以减少计算适应度值所需的时间. 国内也有很多研究团队提出了相关的工作, 例如 Tian 等人[8]提出的 MOSAT 认为 NPC (non-player character) 车辆的自由组合不能

生成有挑战的测试场景, 他们定义了 4 种公共主题模式让 NPC 车辆执行指定的靠近自主车辆的行为. 这些方法专注于生成指定领域的场景, 譬如自动驾驶, 而难以迁移到其他的连续决策场景, 如机器人控制.

除了上述针对自动驾驶系统的测试方法, Pang 等人[9]提出的 MDPFuzz 是一个通用的针对 MDP 连续决策过程的模糊测试框架, 通过变异测试序列的初始状态来生成多样的测试用例. 但 MDPFuzz 是一种端到端的黑盒测试方法, 即使当前测试序列不太可能造成失效, 也必须跑完每一轮测试, 导致发现失效事故的效率较低, 而本文提出的方法旨在解决该问题.

1.3 面向连续决策模型的对抗攻击方法

面向连续决策模型的对抗攻击方法大致可以分为 3 类: 基于观测值的攻击、基于动作/轨迹操纵的攻击和基于对抗智能体的攻击. 在基于观测值的攻击中, Huang 等人[19]提出的代表性方法, 在每个时间步对被测智能体的观测值 (状态) 进行微小的扰动, 以影响智能体的决策和行为, 使其策略网络输出次优行为, 其目的是引导被测智能体出现决策错误. 基于动作的攻击是在智能体动作上进行扰动, 通过操纵被测智能体的行动轨迹, 诱导其发生偏离正常路径等决策错误, 在白盒和黑盒测试中都可应用, 如 Lee 等人[20]提出的针对深度强化学习智能体模型的时空约束动作空间进行攻击. 基于对抗智能体的攻击是训练对抗策略, 如 Gleave 等人[21]训练对抗智能体用于 MuJoCo 游戏的测试. 这类工作主要关注对强化学习智能体的攻击, 与本文的不同之处在于攻击所采用的大多是数字对抗样本 (例如改变图像像素), 这种样本在现实世界中可能并不存在, 且无法揭示模型中存在的策略缺陷.

2 基础知识

2.1 马尔可夫决策过程

马尔可夫决策过程 (MDP) 是连续决策算法 (SDM) 的数学模型, 通过离散时间的随机规划模拟智能体与环境交互可实现的随机策略和可以获得的奖励回报[1,9], 如图 1 所示. MDP 可以被表示为 (S, A, T, R, π) 的元组, 分别表示状态、动作、转移、奖励和策略. 更具体地说, 状态 S 是智能体从环境中观察到的状态 (也称为状态空间). 动作 A 是智能体可以采取的一组动作 (也称为动作空间). 转移 T 是一个函数 $s_{t+1}=T(s_t, a_t)$, 智能体在时间步 t 观察的状态为 s_t, 采取动作 a_t 后, 转移函数 T 使环境转移到下一个状态 s_{t+1}. 奖励 R 是智能体在 t 时间采取动作后得到的即时奖励, 定义为 $r_t = R(s_t, a_t)$, 策略 $\pi(a|s_t)$ 是在 t 时间所有可能动作的概率分布, 以指导智能体在状态 s_t 下采取动作 a_t 以获得最大化的奖励. 正如图 1 所示, 智能体按时间步与环境进行交互, 时间步 $t \in \{0,1,\ldots,M-1\}$. MDP 重复这个过程直到任务成功/失败, 或达到时间步 M 的上限. 当 MDP 完成时, 总的状态序列 $\{s_t\}_{t\in\{0,1,\ldots,M-1\}}$ 和累积奖励 r 反映了智能体的行为模式和能力.

2.2 MDP 的策略学习模型

MDP 的关键是获得一个最优策略 π, 基于深度学习获得最优策略是最受欢迎和有效的解决方案. 从现有文献中可以归纳为 3 种类型.

1) 强化学习 (reinforcement learning, RL) 目的是让智能体学习在奖励的刺激下, 逐步形成对刺激的预期, 从而学习得到能获得最大利益的习惯性行为, 即最优策略. 强化学习模型是解决连续决策问题最常用的模型之一, 其优势在于不需要做数据标注, 而是关注智能体如何基于环境而行动, 以获得 MDP 最大的累积奖励. 近年来, 越来越多的 RL 算法 (例如 DQN[22], A3C[23], PPO[24]等) 在一些复杂场景中表现出超越人类的潜力, 如围棋[25], 电子游戏[26]和机器人控制[27]等.

2) 模仿学习 (imitation learning, IL)[28]的目的是通过监视专家的状态/动作序列 $\tau = (s_0, a_0, s_1, a_1, \ldots)$, 直接从专家演示中学习执行任务的策略. 强化学习的一个困难是很难设计出合适的奖励函数, 而 IL 则让智能体直接模仿专家来解决这个问题. 已有的研究也证明模仿学习在自动驾驶等领域有出色的表现[29,30].

3) 多智能体强化学习 (multi-agent reinforcement learning, MARL)[31]用于处理涉及多个智能体之间交互的场景, 智能体可以合作完成一个任务, 或者彼此竞争形成一个零和马尔可夫博弈. 在 MARL 中, 由于多个智能体采取的动作相互影响并共同影响环境, 智能体共同进化, 所以复杂性极高. MARL 已经在视频游戏[32]和交通控制[33]

等任务中得到应用.

3 干扰惰性序列的模糊测试方法——IIFuzzing

模糊测试虽然通过种子变异力求触发更多的缺陷或者失效事故, 但由于根本上是黑盒测试, 对于连续决策算法这类决策过程长, 环境不确定性很强的智能模型, 初始状态很难影响到后续的决策序列, 其探索失效事故的能力是非常有限的, 实践中通常大多数的测试序列都没有触发失效事故.

为了解决连续决策智能算法在模糊测试时, 未考虑 SDM 决策过程内部的状态, 导致测试耗时且触发失效事故效能不够的问题, 本文提出了通过干预连续决策过程, 预测并中止惰性测试序列的方法——IIFuzzing. 方法的框架如图 2 所示. IIFuzzing 采用经典的模糊测试范式, 包括 4 个主要组件: 1) 仿真测试组件, 基于 MDP 构建仿真测试环境, 并执行测试, 触发事故或执行完一次任务为一轮测试. 2) 模糊测试框架, 包括初始状态生成器和反馈分析算法, 其中初始状态生成器从种子库选择初始状态, 并进行种子变异. 反馈分析算法建立序列评价方法, 识别具有触发失效事故潜力的状态加入种子状态库, 回馈给算法模型学习. 3) 惰性序列预测模型, 基于历史数据构建惰性序列预测模型, 预测并中止惰性序列执行. 4) 干预点设置, 基于实验方法设置合理的干预点, 在控制风险的前提下尽量减少测试资源消耗.

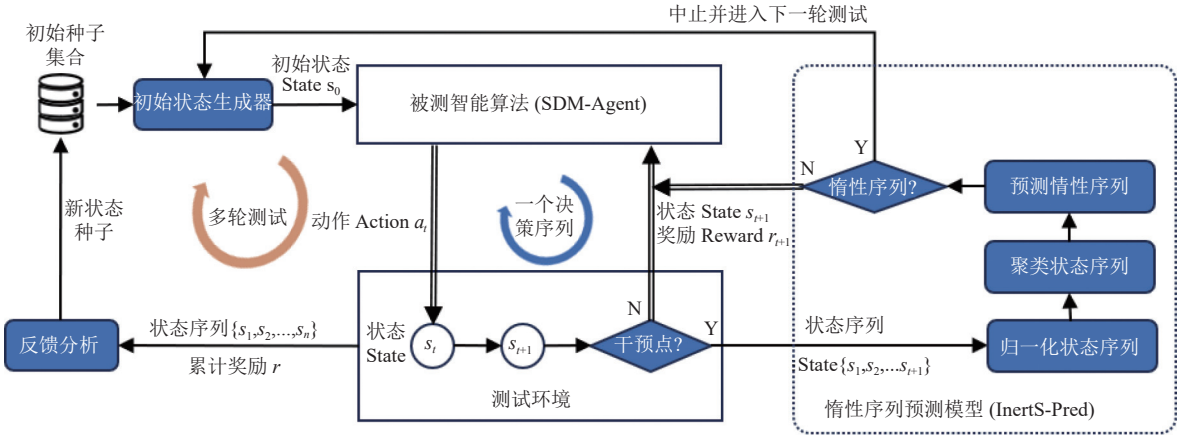


图 2 IIFuzzing 方法的框架

3.1 仿真测试组件

连续决策智能算法的核心是将算法 (SDM-Agent) 和需要交互的环境建模为 MDP, 并利用强化学习、模仿学习等算法基于任务需求学习出最优的决策策略, IIFuzzing 构建待测算法和相关环境的交互仿真组件, 给定测试的初始状态, 被测智能算法根据预定策略和逐步优化的策略执行动作, 进入下一状态并获得奖励, 驱动任务执行直到发生失效事故 (如碰撞)、完成任务或者执行最大时间步 M. 由于不同任务的耗时不同, M 是 IIFuzzing 的超参数, 在测试实践中, 根据测试资源、环境、任务耗时等, 由测试专家根据经验确定.

3.2 模糊测试框架

IIFuzzing 基于通用的模糊测试框架, 主要包括初始状态种子选择、种子变异和反馈分析.

3.2.1 初始状态生成器

初始状态生成器的目的是通过种子选择和变异为测试生成尽可能触发失效事故的初始状态. IIFuzzing 建立和维护一个种子语料库, 语料库的初始值通过在 MDP 的合法状态空间中随机采样初始状态获得, 也可以是历史的种子库. 在测试过程中, 通过反馈分析, 还可以增加新的种子. 借鉴传统软件模糊测试中的“种子能量”(表征该种子的变异次数, 能量越高表明该种子变异次数越多, 反之变异次数越少)[34]的思想, 我们利用“灵敏度”(*Sensitivity*)[9]基于历史数据评估各测试序列的初始状态揭示失效事故的可能性. 具体地, 向初始状态 s_0 (种子) 添加随机扰动 Δs,

收集 MDP 交互后的累积奖励 Δr, 则“灵敏度”的计算公式为:

$$Sensitivity = \frac{|r - \Delta r|}{||\Delta s||_2} \tag{1}$$

其中, r 是不添加扰动时的累积奖励. “灵敏度”越高, 表示所选种子触发失效事故的可能性越高. 每轮测试首先选择尚未测试的“灵敏度”最高的种子, 然后根据原始数据的类型将随机噪声添加到初始状态来施加突变, 例如, 将 $N(0, 1)$ 的高斯浮点数添加到 ADS 环境中车辆的初始位置. 随机变异同时还要满足一些简单的约束, 以确保变异的初始状态是有效的和可解的 (例如, 一开始不会发生事故). 变异后的状态作为初始状态输入待测试的算法模型.

3.2.2 反馈分析算法

反馈分析的目的是识别可能暴露新覆盖模式 (譬如软件测试中的探索路径) 的新种子, 回馈给模糊测试以发现更多的缺陷. 在离散的深度神经网络 (deep neural network, DNN) 的模糊测试中, 通行的方法是度量给定输入在 DNN 中的神经元覆盖度. 但对于连续决策模型, 其中策略学习模型都是黑盒子, 这个方法是不适用的. IIFuzzing 考虑状态序列的差异性和累积奖励, 作为判断某序列的初始状态是否加入种子库的依据. 我们参考 MDPFuzz 提出基于概率密度的状态新鲜度 (Freshness) 度量算法[9]. Freshness 的工作原理是首先根据现有的状态序列估计一个概率密度函数, 然后将新的状态序列与现有序列进行比较, 再根据概率密度函数计算新序列的密度. 如果新序列的密度较低, 则表明它没有被现有序列覆盖. 也就是说, 低密度表示高新鲜度, 即由该变异种子派生的状态序列和之前的状态序列越不同, 也意味着该变异种子应该加入种子库. 此外, 累积奖励越高表明模型完成任务的可能性越高, 相反累积奖励越低则表示该序列触发失效事故的可能性较高, IIFuzzing 综合考虑两个指标将越不同 (高新鲜度) 且越容易触发失效事故 (低累积奖励) 的序列的初始状态作为新种子加入种子库.

在我们的实验环境中, 仅通过上述基础的模糊测试框架生成的测试序列大都不能触发失效事故 (详情见第 4.5.4 节的结果分析). 这是因为在测试过程中 MDP 过程是一个黑盒子, 且初始种子变异具有局限性, 导致无法大量稳定地生成具有挑战性的测试场景. 为解决该问题, 我们提出了惰性序列预测模型.

3.3 惰性序列预测模型

既然很多测试序列都不会触发失效事故, 那么我们是否可以打破 MDP 的黑盒子, 提前预判后续测试动作触发失效事故的可能性, 中止那些不太可能触发失效事故的测试序列 (称为惰性序列) 呢? 为此, IIFuzzing 构建了一个惰性序列预测模型 InertS-Pred. IIFuzzing 设置一个干预时间点 (参见第 3.4 节), 然后在该时间点, InertS-Pred 分析已产生的状态序列, 预测该序列是否为惰性序列. InertS-Pred 主要包括 3 部分.

3.3.1 状态序列归一化表示

从 MDP 得到的状态序列是一种高维数据, 其形式为 $t \times s$ 的多变量时间序列. 由不同长度的多个时间步的状态组成, 每个时间步的状态都是 s 维向量. 我们采用无监督和可扩展的表示学习方法[35,36], 构建了一个可扩张卷积的深度卷积神经网作为本方法的表示学习模型, 该模型可以将不固定长度的高维数据表示转化为固定维数的向量. 我们用数据训练该模型, 在 t 时刻, 将该时刻前的状态序列 $\{s_0, s_1, \ldots, s_{t-1}\}$ 输入训练好的表示学习模型, 得到归一化的向量表示.

3.3.2 基于密度聚类状态序列

显然, 任何一款有质量的软件或者算法, 其成功的概率都是远远大于失效的概率. 对基于 MDP 的连续决策算法, 仿真测试时, 成功完成任务的轮次也是远远大于失效的轮次. 社会学上有一种普遍的观点, 认为“成功大都相似, 失败却各有理由”, 我们的基本假设是: 成功完成任务的状态序列可能呈现相似的模式且聚集在高密度区域, 触发失效事故的序列则呈现明显不同的模式且分布在低密度的外围边界区域. 我们分析了不同测试环境下状态序列在干预点时的密度聚类分布情况, 其结论也很好地印证了上述假设 (详情见第 4.5.4 节的结果分析). 基于此, 我们使用基于密度的状态序列聚类算法, 利用分层密度空间聚类方法 (hierarchical density-based spatial clustering of applications with noise, HDBSCAN)[37]对上一步得到的状态序列的归一化表示进行聚类. 我们基于现有的算法包开发了该模块.

IIFuzzing 的目标是尽可能早地预测出可能触发失效事故的序列. 我们首先利用历史数据 (带有是否触发事故的标签) 训练模型, 通过在不同时间步的历史状态序列数据拟合以确定每个聚类中的最小成员数和最优干预点. 关于干预点选择的细节见第 3.4 节.

拟合后的模型在线工作时, 对于给定的归一化表示的状态序列, 模型将输出一个聚类向量 $(p_1, p_2, \ldots, p_n)$, 该向量的长度等于拟合模型中聚类的数目, 向量中第 i 个数据点 p_i 的值表示该状态序列属于第 i 个聚类的概率. 我们发现可能触发失效事故的序列到任何已有聚类的概率都非常低.

3.3.3 预测惰性序列

惊喜充分性 (surprise adequacy, SA) 是一种度量 DNN 测试充分性的方法[38], 表征新输入的样本相对于训练样本的新颖性. DeepGini[39]进一步将该指标用于选择 DNN 的测试用例. 对于得到的聚类向量, 可能触发失效事故的序列属于任何已有聚类的概率都非常低. 对于给定状态序列, 我们已经得到该序列属于某个聚类的概率向量. 参考 DeepGini 我们将状态序列惰性的计算公式定义为:

$$SA = \sum_{i=1}^{c} p_i^2 \tag{2}$$

SA 越小, 该状态序列触发失效事故的可能性越大. 实践中可以根据测试资源、风险控制水平等要素经验性确定一个阈值, SA 高于该阈值则预测为惰性序列, IIFuzzing 将中止该序列的继续执行.

3.4 干预点设置

干预点的选择对 IIFuzzing 的影响很大, 越早干预并中止惰性序列的执行, 可以节省测试资源, 在相同时间内执行更多的测试用例; 但过早干预也提高了错误预测并中止有效测试的风险. 我们通过实验方法, 在多个间隔时间点综合考虑惰性和非惰性序列预测的召回率 (记为 R_{is}, R_{nis}) 和时间 t 来设置合适的干预点. 具体的选择方法遵循两个准则: 1) 选择惰性和非惰性序列预测的召回率的平均值 $Average(R_{\text{is}}, R_{\text{nis}})$ 最高的时间点作为备选干预点; 2) 如果平均值最高的前两名之间的差距低于 5%, 则选择时间 t 靠前的时间点作为干预点. 更详细的介绍见第 4.5.4 节.

4 实验验证

4.1 实验验证目标

实验设计的目标是验证 IIFuzzing 的有效性. 为此我们设计 4 个评价问题 (research question, RQ).

RQ1: IIFuzzing 探索连续决策模型失效事故的有效性和效率?

RQ2: IIFuzzing 预测惰性和非惰性状态序列的准确性?

RQ3: IIFuzzing 探测到的失效事故的多样性?

RQ4: 干预点设置的合理性?

4.2 测试目标模型与测试环境

如图 1 所示, 连续决策问题通常建模为 MDP 过程, 通过智能体模型和环境的交互执行完成预设任务. 为了验证 IIFuzzing 的有效性, 我们选择并设置了 4 种目标模型和测试环境的组合配置, 摘要信息如表 1 所示.

(1) RL+CARLA: CARLA[40]是一个流行并开源的自动驾驶仿真系统, 包括行人、路口和交通灯, 用于自动驾驶系统在城市仿真道路的训练和验证. 我们选择了一个可在 CARLA 执行的、先进的强化学习模型[41]作为待测的目标智能体. 该模型在 CARLA 挑战赛的“仅用相机”赛道中获胜[42].

(2) IL+CARLA: 同样在 CARLA 的环境上, 我们选择了一个模仿学习模型[30]. 该模型曾经在 CARLA 自动驾驶排行榜排第一[43].

表 1 测试配置

算法模型+测试环境	领域	最大时间步M	状态维度	历史/初始数据	
				测试序列数	失效事故率 (‰)
RL+CARLA	自动驾驶	100	17	714	40.60
IL+CARLA	自动驾驶	200	17	516	50.40
RL+BipedalWalker	机器人控制	300	24	6 148	11.90
MARL+CoopNavi	合作导航	100	24	13 194	0.30

以上两个模型的主要动作都是通过使用车辆摄像头收集的图像作为输入来确定转向和加速度.

(3) RL+BipedalWalker: BipedalWalker[44]是 OpenAI Gym 中的一个双足机器人仿真环境, 包括草地、台阶、坑和树桩等, 我们选择基于强化学习的 TQC 模型[44]作为目标智能体. TQC 以 24 维状态为输入, 根据身体角度、腿部角度、速度和激光雷达等数据预测每条腿的速度, 引导智能体穿过草地、树枝等障碍. 该模型的实现开源在著名的 stablebaseline3 算法包内.

(4) MARL+CoopNavi: CoopNavi[45]是 OpenAI 发布的一个合作导航环境, 用于多智能体合作在没有碰撞的情况下到达一组地标, 是一个多智能体的强化学习环境. 我们使用现有工作[45]中公开的方法构建 MARL 模型, 通过观测其他智能体和地标的相对位置选择移动方向和速度.

4.3 基线模型和评价指标

4.3.1 基线模型

对连续决策模型的测试是一个非常有挑战性的方向, 目前大多数方法的目标是生成安全攸关的场景, 例如 AV-FUZZER 和 MOSAT 让自动驾驶车辆在生成的场景环境中仿真执行以触发事故, 但这些场景都是与被测系统的领域强相关的, 即自动驾驶的场景无法迁移到机器人行走的场景. 如前所述, 连续决策问题通常可以建模为 MDP 过程, MDPFuzz 中提出了一个通用的模糊测试框架, 能够对不同领域的连续决策模型进行测试. 我们的工作是基于模糊测试基础框架, 通过预测不会触发失效事故的测试序列, 提前中止惰性的测试序列, 以期在相同的时间内, 可以执行更多的测试序列, 并触发更多且有差异性的失效事故. 基于上述考虑, 我们选择 MDPFuzz 作为对比基线.

4.3.2 评价指标

首先, 我们首先定义失效事故, 亦即测试断言. 由于不同的实验系统任务目标不同, 因此任务失败 (即失效事故) 的定义存在差异. 对于自动驾驶环境 CARLA, 失效事故定义为“自主车辆 (ego-vehicle) 与其他车辆 (NPC-vehicle) 或者障碍物发生碰撞”; 对于机器人行走 BipedalWalker, 失效事故定义为“自走智能体 (ego-walking agent) 跌倒”; 对于合作导航 CoopNavi, 失效事故定义为多智能体之间发生碰撞. 同时, 失效事故也可以根据用户关心的不同方面或测试目标进行定义, 例如在自动驾驶系统中, 我们不仅可以将车辆碰撞定义为失效事故, 还可以将车辆压线、闯红灯等违反交通规则的行为定义为失效事故.

评价指标包括:

(1) 为了评价 IIFuzzing 的有效性和效率, 我们采用两个最常用的指标:

- #Crash: 表示在指定时间内触发的失效事故的数量.
- #Test: 表示在指定时间内执行的测试轮次.

(2) 为了评价 IIFuzzing 预测惰性序列的准确度, 我们采用了分类模型中最常用的两个指标:

- Precision: 表示正确识别的阳性标签占模型预测的阳性标签的比例.
- Recall: 表示正确识别的阳性标签占样本中总阳性标签的比例.

4.4 实验设置

IIFuzzing 用 Python 编写, 通过 MDP 框架和目标模型及环境交互. 所有被测目标模型运行在 PyTorch 环境. 我们根据各模型在测试环境中状态变化的速度, 为上述 4 种测试配置分别设置合适的最大执行时间步 M. 对于输入的初始状态, 模型在执行中如果没有触发失效事故, 且未被预测为惰性序列, 则执行 M 步后即视为完成任务, 执行下一轮测试.

表 1 给出了 4 种测试配置的具体信息. 其中“历史/初始数据”栏表示该测试配置中的历史数据, 或者先随机执行生成的初始数据. 在本实验中, 我们先让模型在配置环境中随机运行 2 h, 将产生的初始状态空间作为初始种子状态集合. 然后在不干预的情况下运行 12 h, 获得数据作为“历史/初始数据”用来训练状态序列表示学习模型和聚类模型, 以及确定最优干预点. 从这里我们可以看到, 在不干预的情况, MDP 的模糊测试只能触发非常少的失效事故.

所有实验都是在配备 NVIDIA TITAN RTX GPU、Intel Xeon Silver CPU、64 GB RAM 的服务器上进行的, 所使用的操作系统为 Ubuntu 20.04.

对于 RQ1 的实验, IIFuzzing 和基线方法 (MDPFuzz) 按照相同的标准测试配置, 即每个配置执行 3 次测试, 每次 12 h[9,46], 本文将汇报 3 次测试的平均结果.

对于 RQ2 的实验, IIFuzzing 在干预点预测到惰性序列后, 并不中止执行, 而是继续完成该轮 MDP 过程, 这样我们知道被预测为惰性序列的测试, 后续是否真实触发了失效事故, 以此来评价 IIFuzzing 惰性预测模型 (InertS-Pred) 的准确性.

对于 RQ3 的实验, 我们首先将 12 h 内所有的状态序列用表示学习模型做归一化表示 (参见第 3.3.1 节), 然后将得到的表示向量送给高斯混合模型 (Gaussian mixture model, GMM)[47]去拟合状态序列的分布, 最后, 我们可视化 GMM 的分布, 通过比较覆盖区域, 来评估 IIFuzzing 和基线方法所执行的测试用例的多样性.

对于 RQ4 的实验, 对每个测试配置, 以其 $M/5$ 作为间隔步长, 设置不同的干预点. 由于主要评价干预点对于惰性预测准确性的影响, 所以和 RQ2 的实验类似, 实验中在干预点只做预测, 并不中止序列执行, 然后通过实际触发的失效事故数据计算每个干预点的双侧召回率, 按照第 3.4 节的方法选择干预点, 然后分析在干预点测试状态序列的分布.

4.5 实验结果与分析

4.5.1 IIFuzzing 有效性和效率 (RQ1)

图 3 展示了 IIFuzzing 和基线方法在 4 个测试配置下随时间执行测试序列和触发失效事故的趋势. 可以看出随时间进展, 在 4 个测试配置下, IIFuzzing 执行测试序列的数量和触发失效事故的数量与基线方法相比明显拉开距离, 时间越长, 优势越明显.

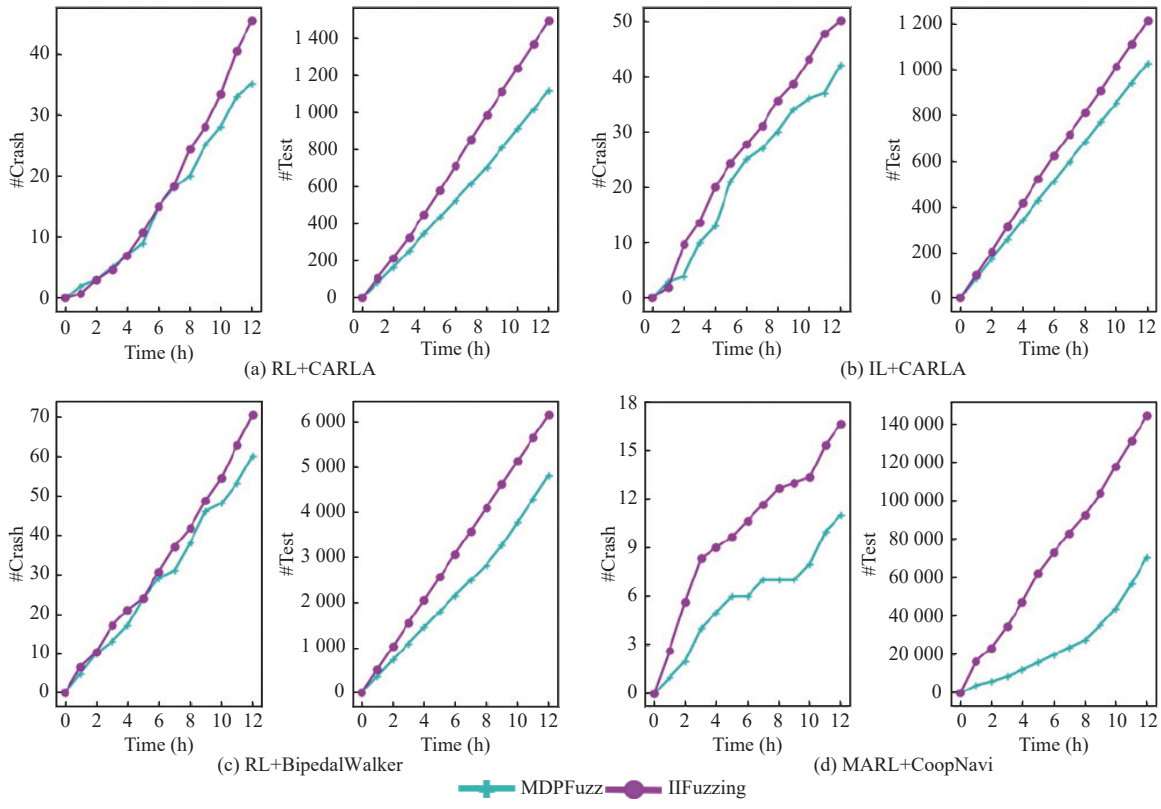


图 3 IIFuzzing 和基线方法执行的测试序列和触发失效事故的效率

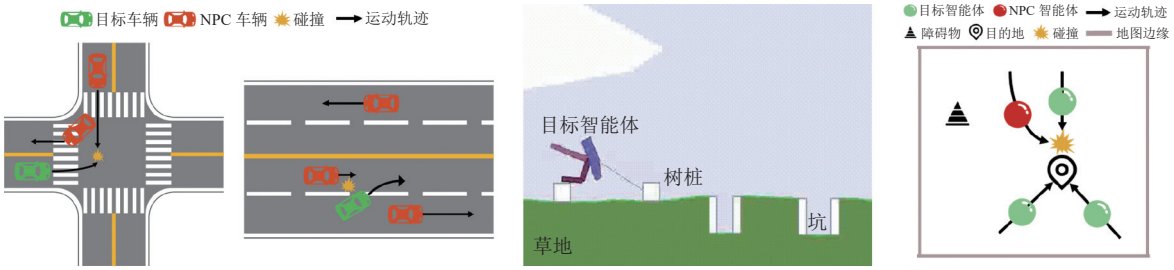
表 2 给出了 3 次 12 h 执行的测试序列的总数和触发失效事故的平均值, 其中#Test 表示执行的测试序列数, #Crash 表示触发的失效事故数. 可以看出, IIFuzzing 在 4 个测试配置中, 可以多执行 18.1%–103.8% 的测试序列,

并多触发 16.7%–54.5% 的失效事故. 在 MARL+CoopNavi 配置中, IIFuzzing 的优势最为明显, 这是因为该测试配置中, 失效事故序列的比例非常低, 所以 IIFuzzing 中止不能触发失效事故的惰性序列, 对测试效能的提升就更加明显. 我们同时也观察到, 在 4 种测试配置中, IIFuzzing 所触发的失效事故的比率并没有明显改变, 这也是一个非常有意义的结果, 说明 IIFuzzing 并非盲目追求执行更多的测试序列, 而是在遵循被测模型缺陷密度自然属性的前提下, 尽可能多快好省地揭示那些引发失效事故的缺陷.

表 2　IIFuzzing 和基线方法的测试性能

测试方法	RL+CARLA		IL+CARLA		RL+BipedalWalker		MARL+CoopNavi	
	#Test	#Crash	#Test	#Crash	#Test	#Crash	#Test	#Crash
MDPFuzz	1 117	35	1 023	42	4 806	60	70 664	11
IIFuzzing	1 491 **(+33.5%)**	45 **(+28.6%)**	1 208 **(+18.1%)**	50 **(+19.0%)**	6 143 **(+27.8%)**	70 **(+16.7%)**	144 016 **(+103.8%)**	17 **(+54.5%)**

图 4 展示了 IIFuzzing 在不同测试配置下找到的独特失效事故的示例. 例如图 4(a) 展示的是在自动驾驶任务中, 目标车辆在十字路口试图左转, 但由于纵向车道先行右转的 NPC 车辆阻碍了视线, 导致目标车辆未能发现紧随其后的试图直行的 NPC 车辆, 从而发生了碰撞; 图 4(b) 展示的同样是在自动驾驶任务中, 目标车辆在变道超车时, 由于相邻车道的两个 NPC 车辆运行速度不稳定, 导致预留的超车空间狭小, 最终导致目标车辆与后车发生碰撞; 图 4(c) 展示的则是控制双足机器人穿越障碍到达终点的任务, 由于生成的场景包含连续的树桩以及不同大小的坑, 复杂的场景导致目标智能体摔倒; 图 4(d) 展示的是在合作导航任务中, 3 个目标智能体试图到达目的地时, 由于 NPC 智能体的突然转向导致与其中之一发生碰撞. 上述这些测试场景由于涉及的环境因素 (障碍物、NPC 智能体) 较多, 对目标智能体的挑战性更大, 但往往需要经过多次探索才有可能生成. 相较于 MDPFuzz, IIFuzzing 的工作原理在于尽可能跳过惰性序列 (非失效序列), 而对非惰性序列 (失效序列) 不进行操作, 从而节省测试资源进行更广的探索. 正因如此 IIFuzzing 在发现失效事故的严重程度上不存在明显差异, 而是在发现失效事故的数量和多样性上占据明显优势.



(a) RL+CARLA　(b) IL+CARLA　(c) RL+BipedalWalker　(d) MARL+CoopNavi

图 4　IIFuzzing 发现的失效事故示例

问题 1 评价结论: IIFuzzing 在同样的测试时间内, 可以执行更多的测试序列, 并触发更多的失效事故. 且其优势随测试时间增加更加明显.

4.5.2　IIFuzzing 预测惰性状态序列的准确性 (RQ2)

IIFuzzing 的性能关键依赖其预测惰性状态序列的准确性, 从表 1 我们可以看到, 仿真测试中, 能触发失效事故的测试序列非常稀疏, 数据极不平衡, 只考虑预测模型的单边精确率和召回率是不合适的. 同时, IIFuzzing 的目标是在尽可能不损失缺陷探测能力的前提下中止惰性序列的执行. 所以, 对于 IIFuzzing 的惰性预测模型 InertS-Pred, 我们首先追求惰性序列预测的准确率, 放宽其召回率. 也就是说宁可放过惰性序列继续执行消耗资源, 也减少误报带来的中止了有可能触发事故的测试序列执行的风险. 在非惰性序列一侧, InertS-Pred 则期望尽可能高的召回率, 亦即期望非惰性序列不会被提前中止, 以免损害探测失效事故的能力. 表 3 展示了 InertS-Pred@IIFuzzing 的实验结果. 我们可以看到在 4 个测试配置中, 有 2 测试配置的惰性序列预测准确率都达到了 100%, 其他也在 97% 以上, 非惰性序列的召回率在 66.7%–100%, 特别在两个强化学习算法中, 可以达到 100% 的惰性序列预测准确率和

100% 的非惰性召回率. 这意味着强化学习得到的策略可以让测试序列的密度分布更加明显, 应用本方法的效果更好. 此外, 我们也可以观察到, 在 MARL+CoopNavi, 惰性序列的召回率也到达 42.5%, 这也进一步解释了为什么在 RQ1 中该测试配置的测试性能提升的最多.

表 3　惰性序列预测模型的性能 (%)

测试配置	P_{is}	R_{is}	P_{nis}	R_{nis}
RL+CARLA	100.0	13.1	4.3	100.0
IL+CARLA	97.6	25.5	4.7	85.7
RL+BipedalWalker	100.0	2.8	1.3	100.0
MARL+CoopNavi	99.8	42.5	0.1	66.7

注: P表示准确率, R表示召回率, 下标is, nis分别表示惰性和非惰性序列

问题 2 评价结论: IIFuzzing 的惰性序列预测模型 InertS-Pred 可以到达很好的惰性序列预测准确率, 同时非惰性序列的预测召回率在 66.7%–100%, 平均可达 88%. 这意味着 IIFuzzing 放行非惰性序列的风险较小, 可以准确跳过不太可能触发失效事故的测试序列, 从而在同一测试周期中留下更多的测试资源来探索更大的测试用例空间.

4.5.3　IIFuzzing 探测到的失效事故的多样性 (RQ3)

IIFuzzing 测试的目标不仅要探测尽可能多的失效事故, 还希望这些事故具有更好的差异性, 也就是能触发更多样的事故. 我们用高斯混合模型 (GMM) 来估计触发了失效事故的所有测试状态序列的分布密度. 分布密度越广, 说明测试序列越具有多样性. 图 5 是 IIFuzzing 和基线方法 MDPFuzz 探测的失效事故序列的 GMM 分布分别在二维平面的投影, 投影面积可视化表示分布密度, 面积越大, 表示分布密度越广, 亦表征估算对象越具有多样性.

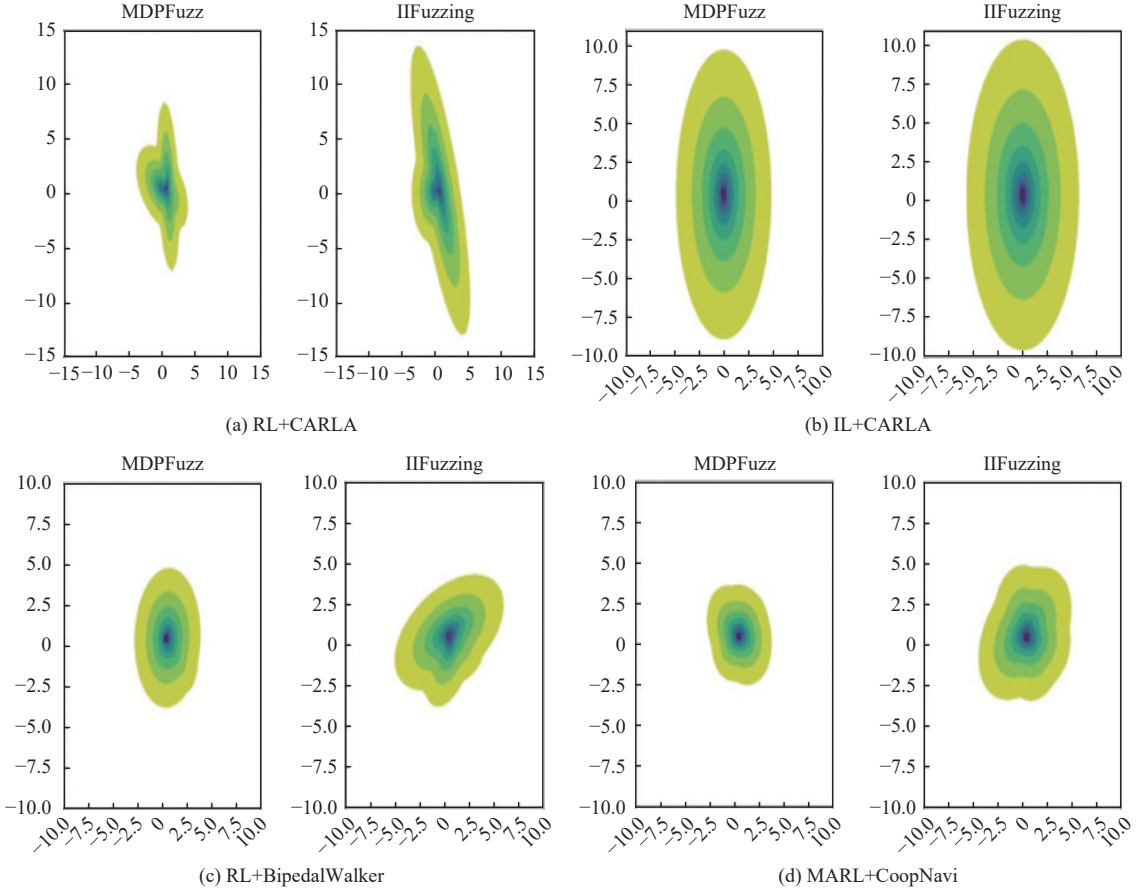


图 5　IIFuzzing 和基线方法探测到的失效事故序列的密度分布

在每个测试配置中, 图 5 的投影图绘制尺度一致, 越趋近蓝色的区域, 表示分布密度越高, 表示这部分的样本越趋同. 反之, 越趋近黄色的区域, 表示分布密度越低, 也表示这部分样本的差异性越大 (亦即越多样). 趋黄色的区域越大, 表示整体样本的多样性越好. 从图 5 可以看出, IIFuzzing 的 GMM 投影面积和黄色区域面积都大于基线方法, 在 RL+CARLA 和 MARL+CoopNavi 测试配置中更加明显, 说明 IIFuzzing 可以探测到更多样的失效事故.

问题 3 评价结论: IIFuzzing 在同样的测试时间内, 可以探索更大的测试空间, 并触发更多样的失效事故.

4.5.4 干预点设置的合理性 (RQ4)

第 3.4 节我们介绍了设置干预点的方法. 由于数据非常不均衡, 我们要兼顾预测双侧的召回率, 期望在尽可能放行非惰性序列的前提下, 减少惰性序列的放行执行, 以减少测试资源消耗. 具体地, 在每个测试配置中, 分别以其最大时间步长 M (见表 1) 的 20% (亦即 20%M) 为起点, 每增加 20% 为间隔, 共设 5 个观测点. 图 6 详细展示了不同干预点 InertS-Pred@IIFuzzing 预测双侧的召回率.

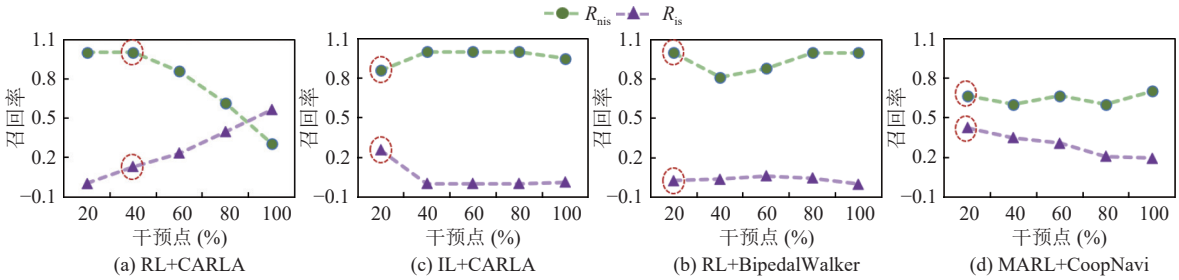


图 6 IIFuzzing 不同干预点的惰性预测召回率

由于 4 个测试配置的差异很大, 我们可以看到惰性序列预测性能随时间进展并没有一致性的趋势, 按照第 3.4 节介绍的方法, 基于实验结果, 红圈标记的点是本方法选择的干预点, 其中 RL+BipedalWalker, 双侧召回率平均值最高的点在 80% 处为 55%), 但在 20% 处双侧的平均召回率为 51.4%, 相差在 5% 以内, 所以, 我们选择靠前的时间点 20% 处作为干预点, 其他 3 个测试配置中, 都是选择的双侧平均最高点.

为了验证干预点的合理性, 我们分析在干预点的状态序列分布. 对所有执行到干预点的测试序列, 基于第 3.3 节介绍的聚类模型做聚类, 然后将聚类后的分布向量输入算法 t-SNE[48]做降维处理并可视化展示在二维平面, 图 7 分别展示了各测试配置中的分布情况. 其中不同的颜色表示聚类后的不同类簇, 特别地, 灰色表示离群点, 亦即认为可能会触发失效事故的状态簇; 圆点表示最终未触发失效事故的测试序列, 即惰性序列, 三角表示最终实际触发失效事故的测试序列, 即非惰性序列. 我们可以看到: 1) 在干预点, 大多数的测试序列为惰性序列; 2) 惰性状态序列通常分布在簇中心的高密度区域, 而非惰性序列则分布在外围的低密度区域; 3) IIFuzzing 识别出的离群点 (即灰色标记) 确实可以定位到分布于外围的低密度区域, 说明干预点的设置是合理的.

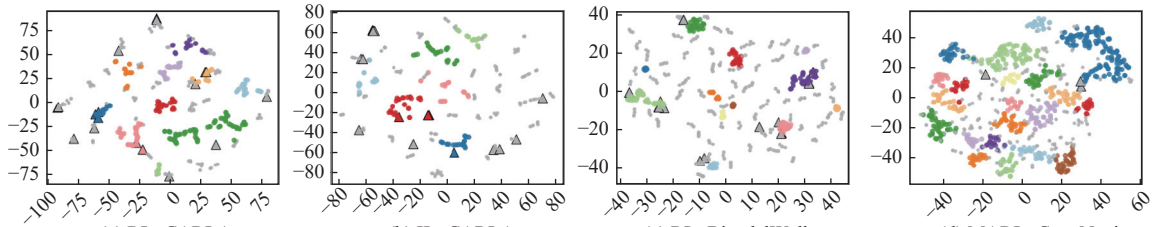


图 7 在干预点的状态序列密度聚类分布

问题 4 评价结论: 测试状态序列在干预点已经呈现比较清晰的密度分布, 惰性和非惰性序列分布于明显不同的聚类, IIFuzzing 干预点的选择方法是合理的.

5 方法有效性威胁分析

5.1 外部有效性

外部有效性威胁涉及所提出方法的通用性. IIFuzzing 的外部有效性威胁主要来自预测惰性序列时要依赖从历史数据中学习到的知识 (例如, 状态序列的表示学习, 聚类的拟合以及干预点的选择等), 这些知识与被测算法和测试环境都是相关的. 当面对新的测试场景时, 需要依据领域相关数据对 IIFuzzing 的惰性预测模型进行训练或者调校. 我们在 3 个仿真环境的 4 种不同类型的模型上进行了实验评估, 所选的目标模型都是当前最先进的模型, 仿真环境也是学术界和工业界广泛应用的环境, 每个模糊测试周期的时间跨度设置为 12 h, 这是实验评估的标准设置[9,46]. 这些测试配置下, 根据第 4.5.1 节 RQ1 中图 3 展示的结果, 可以发现 IIFuzzing 基本上在 12 h 的测试全时段都有性能优势 (#Crash), 且展现出优势不断拉大的趋势, 说明本方法的外部威胁较小, 外部有效性是可控的.

5.2 内部有效性

内部有效性威胁主要集中在 IIFuzzing 的工作机理上. 一方面, 当被测系统存在较多问题时, 测试序列的密度分布情况可能并不满足本文假设, 从而导致现有的方法设计无法准确识别惰性序列进而造成方法失效. 但如果通过分析能够发现该情形下惰性序列新的分布模式, 那么通过调整 IIFuzzing 识别惰性序列的基本原理, 方法依然能够应用. 另一方面, 由于在推断一个测试用例是否应该继续执行或跳过时, IIFuzzing 会引入额外的开销, 如果单个测试序列的执行时间比引入的开销还要短, IIFuzzing 减少测试资源消耗的假设将受到质疑. 据我们的观察和实验, 在连续决策场景中 MDP 的执行通常是耗时的, 以 MARL+CoopNavi 为例, 这是本实验中 12 h 内执行的测试轮次最多的配置, 也意味每一轮测试的时间最短, 该配置每轮测试的时间平均 611 ms, 而 IIFuzzing 惰性序列在线预测的开销仅在毫秒级别, 这说明 IIFuzzing 工作机理是有效的. 此外, 我们用 MDPFuzz 方法提供的源代码来实现基线模型, 以确保实验结果的正确性和公平性. 另一个限制是 IIFuzzing 通过变异初始状态来生成多样的测试用例, 无法在中间状态诱导场景中的元素以生成临界案例或意外情况. 通过变异任意中间状态来生成逼真而具有挑战性的场景需要进一步探索.

5.3 结论有效性

结论有效性的威胁在于评估指标. 我们使用易于理解的指标“#Crash”和“#Test”. 为了回答 RQ2, 我们还使用准确度和召回率来评估识别 (非) 惰性状态序列的性能, 并分别呈现两个标签的双侧预测结果.

6 总 结

连续决策模型由于需要不断地和环境进行“交互-行动”的连续决策, 环境和交互的不确定性使得经典测试方法追求的测试覆盖和缺陷探测效率面临极大的挑战. 模糊测试技术的基本原理是通过向目标系统提供非预期的输入并监控执行结果, 以发现目标系统的缺陷. 但针对连续决策模型, 要执行的是“状态-行动”决策序列, 初始状态很难影响到后续的决策序列. 模糊测试如只对初始状态 (测试输入) 做变异, 是很难触发决策缺陷的, 这也是模糊测试对连续决策模型效能不高的原因. 本文提出的 IIFuzzing, 基于表示学习和密度聚类可以准确预测模糊测试中不会触发失效事故的惰性序列, 并进行自动干预, 从而扩大了资源限制条件下的测试探索空间, 并加速了失效事故的探测效率.

下一步的工作可以基于对决策序列模式的深度学习, 尝试合理干预决策序列的中间状态, 使其导向可能的事故, 以期更主动和更充分地揭示模型的决策缺陷.

References:

[1] Markov decision process. 2024. https://en.wikipedia.org/wiki/Markov_decision_process.

[2] California DMV. Autonomous vehicle collision reports. 2024. https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-collision-reports/.

[3] Guo JM, Jiang Y, Zhao Y, Chen Q, Sun JG. DLFuzz: Differential fuzzing testing of deep learning systems. In: Proc. of the 26th ACM

Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Lake Buena Vista: ACM, 2018. 739–743. [doi: 10.1145/3236024.3264835]

[4] Ma L, Zhang FY, Sun JY, Xue MH, Li B, Juefei-Xu F, Xie C, Li L, Liu Y, Zhao JJ, Wang YD. DeepMutation: Mutation testing of deep learning systems. In: Proc. of the 29th IEEE Int'l Symp. on Software Reliability Engineering. Memphis: IEEE, 2018. 100–111. [doi: 10.1109/ISSRE.2018.00021]

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

[6] Zhang MS, Zhang YQ, Zhang LM, Liu C, Khurshid S. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Montpellier: ACM, 2018. 132–142. [doi: 10.1145/3238147.3238187]

[7] Li GP, Li YR, Jha S, Tsai T, Sullivan M, Hari SKS, Kalbarczyk Z, Iyer RK. AV-FUZZER: Finding safety violations in autonomous driving systems. In: Proc. of the 31st IEEE Int'l Symp. on Software Reliability Engineering. Coimbra: IEEE, 2020. 25–36. [doi: 10.1109/ISSRE5003.2020.00012]

[8] Tian HX, Jiang Y, Wu GQ, Yan JR, Wei J, Chen W, Li S, Ye D. MOSAT: Finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 94–106. [doi: 10.1145/3540250.3549100]

[9] Pang Q, Yuan YY, Wang S. MDPFuzz: Testing models solving Markov decision processes. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 378–390. [doi: 10.1145/3533767.3534388]

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

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

[12] 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. Melbourne: IEEE, 2021. 104–116. [doi: 10.1109/ASE51524.2021.9678670]

[13] 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, 2022. 71. [doi: 10.1145/3551349.3556953]

[14] 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, 2022. 81. [doi: 10.1145/3551349.3556929]

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

[16] Zhou HS, Li W, Kong ZL, Guo JF, Zhang YQ, Yu B, Zhang L, Liu C. DeepBillboard: Systematic physical-world testing of autonomous driving systems. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 347–358.

[17] Gambi A, Mueller M, Fraser G. AsFault: Testing self-driving car software using search-based procedural content generation. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. Montreal: IEEE, 2019. 27–30. [doi: 10.1109/ICSE-Companion.2019.00030]

[18] Abdessalem RB, Panichella A, Nejati S, Briand LC, Stifter T. Testing autonomous cars for feature interaction failures using many-objective search. In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 143–154. [doi: 10.1145/3238147.3238192]

[19] Huang SH, Papernot N, Goodfellow IJ, Duan Y, Abbeel P. Adversarial attacks on neural network policies. In: Proc. of the 5th Int'l Conf. on Learning Representations. Toulon: OpenReview.net, 2017.

[20] Lee XY, Ghadai S, Tan KL, Hegde C, Sarkar S. Spatiotemporally constrained action space attacks on deep reinforcement learning agents. In: Proc. of the 34th AAAI Conf. on Artificial Intelligence. New York: AAAI, 2020. 4577–4584. [doi: 10.1609/aaai.v34i04.5887]

[21] Gleave A, Dennis M, Wild C, Kant N, Levine S, Russell S. Adversarial policies: Attacking deep reinforcement learning. In: Proc. of the 8th Int'l Conf. on Learning Representations. Addis Ababa: OpenReview.net, 2020.

[22] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing Atari with deep reinforcement learning. arXiv:1312.5602, 2013.

[23] Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In: Proc. of the 33rd Int'l Conf. on Machine Learning. New York City: JMLR, 2016. 1928–1937.

[24] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv:1707.06347, 2017.

[25] Silver D, Huang A, Maddison CJ, *et al*. Mastering the game of go with deep neural networks and tree search. Nature, 2016, 529(7587): 484–489. [doi: 10.1038/nature16961]

[26] Berner C, Brockman G, Chan B, *et al*. Dota 2 with large scale deep reinforcement learning. arXiv:1912.06680, 2019.

[27] Kober J, Bagnell JA, Peters J. Reinforcement learning in robotics: A survey. The Int'l Journal of Robotics Research, 2013, 32(11): 1238–1274. [doi: 10.1177/0278364913495721]

[28] Ho J, Ermon S. Generative adversarial imitation learning. In: Proc. of the 30th Int'l Conf. on Neural Information Processing Systems. Barcelona: Curran Associates Inc., 2016. 4572–4580.
[29] Shin M, Kim J. Randomized adversarial imitation learning for autonomous driving. In: Proc. of the 28th Int'l Joint Conf. on Artificial Intelligence. Macao: ijcai.org, 2019. 4590–4596. [doi: 10.24963/ijcai.2019/638]
[30] Chen D, Zhou B, Koltun V, Krähenbuhl P. Learning by cheating. In: Proc. of the 3rd Annual Conf. on Robot Learning. Osaka: PMLR, 2019. 66–75.
[31] Busoniu L, Babuska R, De Schutter B. A comprehensive survey of multiagent reinforcement learning. IEEE Trans. on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2008, 38(2): 156–172. [doi: 10.1109/TSMCC.2007.913919]
[32] Vinyals O, Babuschkin I, Czarnecki WM, *et al*. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 2019, 575(7782): 350–354. [doi: 10.1038/s41586-019-1724-z]
[33] Wiering M. Multi-agent reinforcement leraning for traffic light control. In: Proc. of the 17th Int'l Conf. on Machine Learning. Stanford: Morgan Kaufmann Publishers Inc., 2000. 1151–1158.
[34] Bohme M, Pham VT, Roychoudhury A. Coverage-based greybox fuzzing as Markov chain. IEEE Trans. on Software Engineering, 2019, 45(5): 489–506. [doi: 10.1109/TSE.2017.2785841]
[35] Franceschi JY, Dieuleveut A, Jaggi M. Unsupervised scalable representation learning for multivariate time series. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 418.
[36] van den Oord A, Dieleman S, Zen HG, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A, Kavukcuoglu K. WaveNet: A generative model for raw audio. In: Proc. of the 9th ISCA Speech Synthesis Workshop. Sunnyvale: ISCA, 2016. 125.
[37] McInnes L, Healy J, Astels S. hdbscan: Hierarchical density based clustering. Journal of Open Source Software, 2017, 2(11): 205. [doi: 10.21105/joss.00205]
[38] 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]
[39] 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]
[40] Dosovitskiy A, Ros G, Codevilla F, Lopez A, Koltun V. CARLA: An open urban driving simulator. In: Proc. of the 1st Annual Conf. on Robot Learning. Mountain View: PMLR, 2017. 1–16.
[41] Toromanoff M, Wirbel E, Moutarde F. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In: Proc. of the 2020 IEEE/CVF Conf. on Computer Vision and Pattern Recognition. Seattle: IEEE, 2020. 7151–7160. [doi: 10.1109/CVPR42600.2020.00718]
[42] The CARLA autonomous driving challenge. 2024. https://carlachallenge.org/
[43] CARLA autonomous driving leaderboard. 2024. https://leaderboard.carla.org/leaderboard/
[44] Kuznetsov A, Shvechikov P, Grishin A, Vetrov D. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In: Proc. of the 37th Int'l Conf. on Machine Learning. PMLR, 2020. 5556–5566.
[45] Lowe R, Wu Y, Tamar A, Harb J, Abbeel P, Mordatch I. Multi-agent actor-critic for mixed cooperative-competitive environments. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6382–6393.
[46] Klees G, Ruef A, Cooper B, Wei SY, Hicks M. Evaluating fuzz testing. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 2123–2138. [doi: 10.1145/3243734.3243804]
[47] McLachlan GJ, Basford KE. Mixture Models: Inference and Applications to Clustering. New York: Marcel Dekker, 1988.
[48] van der Maaten L, Hinton G. Visualizing data using t-SNE. Journal of Machine Learning Research, 2008, 9(86): 2579–2605.

吴泊逾(1990－), 男, 博士生, 主要研究领域为自动驾驶系统测试.

王亚文(1993－), 男, 博士, 助理研究员, 主要研究领域为智能软件测试, 智能模型对抗攻击.

王凯锐(1999－), 男, 硕士, 主要研究领域为智能软件工程, 智能体测试.

王俊杰(1987－), 女, 博士, 研究员, CCF 专业会员, 主要研究领域为智能软件工程, 软件工程大数据, 经验软件工程, 软件质量, 众包软件测试.