基于真实故障的深度强化学习变异测试

Florian Tambon[§],Vahid Majdinasab[§], Amin Nikanjam, Foutse Khomh, Giuliano Antoniol
蒙特利尔理工学院´，加拿大{florian-2.tambon, vahid.majdinasab,
amin.nikanjam,foutse.khomh, giuliano.antoniol}@polymtl.ca

arXiv:2301.05651v1 [cs.LG] 13 Jan 2023

摘要—**深度学习（DL）系统的测试是一项复杂任务，因为它们的行为与传统系统不同，尤其是其具有随机性。尽管如此，若能将变异测试（MT）等现有测试技术应用于深度学习环境，将大幅提升其可验证性。虽然已有一些研究尝试将变异测试扩展至监督学习范式，但将其扩展至强化学习（RL）的研究仍然很少。强化学习也是深度学习生态系统的重要组成部分，但其行为与监督学习截然不同。本文以现有的变异测试方法为基础，提出一个适用于强化学习变异测试的框架*RLMutation*。具体而言，我们利用现有的故障分类体系，构建一组适用于强化学习的变异算子，并采用简单的启发式方法为强化学习生成测试用例。借此，我们能够基于现有方法比较不同的变异体杀死定义，并分析所得变异算子的行为及其潜在组合——高阶变异（HOM）。我们表明，变异杀死定义的设计选择会影响变异体是否被杀死以及生成的测试用例。此外，我们发现，即使测试用例和变异算子的数量相对较少，我们仍能生成具有有趣特性的高阶变异，从而增强强化学习系统的测试能力。**

索引词—**强化学习、深度学习、变异测试、真实故障**

I. 引言

变异测试是一种白盒测试方法，旨在基于真实故障注入人为变更，以评估测试套件发现故障的能力。变异测试已在传统软件工程中得到广泛研究和应用 [1], [2]，用于评估测试套件的质量。变异测试的一个基本假设是耦合效应假设，该假设认为：“复杂变异体与简单变异体相耦合，因而能够检测程序中所有简单变异体的一组测试用例，也能检测出**很大比例**的复杂变异体” [3]。因此，揭示此类未被杀死的复杂变异体值得关注；生成这类复杂变异体的一种方法是将简单变异体（称为一阶变异）组合起来。这就是 Jia 等人提出的高阶变异概念 [4]。

此类成熟的测试技术和概念或许有助于深度学习系统提高其可靠性，因为这类系统具有独特的性质，众所周知很难进行测试 [5]。由于这种范式使基于深度学习的系统与传统软件系统存在差异，研究人员直到最近才开始提出专为基于深度学习的系统设计的变异测试框架，尤其针对监督学习 [6]–[8]，，以评估测试数据集发现故障的质量。

然而，监督学习并非机器学习中唯一的子范式；（深度）强化学习（RL）是另一种主要子范式，应用范围广泛， [9], [10] 如今也越来越多地应用于实践。强化学习与监督学习有着本质区别：监督学习通过训练数据集学习模型，使其能够泛化到来自输入分布的任意新数据；而强化学习的核心思想是通过反馈系统，让智能体与环境交互来进行训练 [11]。例如，一个机器人（智能体）在房间（环境）中活动，目标是从 A 到 B，途中还设有一些陷阱。因此，先前提出的框架若应用于强化学习，可能会面临若干局限。例如，用于监督学习、以生成变异体模型的变异算子，可能并不适用于强化学习。据我们所知， [12] 是唯一一项针对强化学习中的变异测试展开研究的工作；该研究提出了一种基于人工构建相关环境的故障检测方法。具体而言，他们提出，可以将传统软件系统变异测试中使用的传统测试用例，转化为强化学习中的测试环境。然而，他们的研究仅限于一种强化学习算法，既未探索基于真实故障的算子，也未探讨组合现有算子形成高阶变异（HOM）的潜在价值，而这种方法或可用于评估测试环境发现强化学习系统中细微故障的能力。

本文提出了一个框架 *RLMutation*，用于针对深度强化学习程序进行变异测试，并采用适用于强化学习的高阶变异。我们根据现有的分类故障，为强化学习定义了变异算子。随后，我们采用并比较了若干源自先前研究且针对强化学习环境和算法调整的变异体杀死定义，分析这些定义在不同强化学习环境和算法中的表现。为了利用高阶变异的能力来揭示更复杂的故障，我们将现有的高阶变异研究 [4] 适配到强化学习任务的特性中。具体而言，我们设计了一种针对强化学习问题的简单启发式方法，以系统地生成一些测试环境，进而获得测试用例，用于评估高阶变异的实用性。因此，我们旨在探讨如何将变异测试应用于强化学习，

[§]共同贡献

同时承认强化学习与监督学习之间存在的差异。

我们的贡献是提出了一个强化学习框架，其中包括：基于真实分类故障的 11 种变异算子；对变异杀死定义的设计对被杀死的一阶变异的影响进行比较；以及一种用于生成相关测试环境的启发式方法，以研究一阶变异和高阶变异。

本文其余部分安排如下：第II节介绍变异测试、高阶变异和强化学习的相关背景知识。第三节介绍所提出的变异算子，以及在强化学习的变异测试中确定如何生成相关高阶变异的流程。第四节报告实验和结果。第五节讨论本研究的有效性威胁。第六节回顾相关文献，第七节总结全文。

II. 背景

A. 强化学习

强化学习是一种机器学习子范式，智能体通过与环境 [11]交互来学习。其主要任务是学习如何将状态映射到动作，同时最大化长期奖励。因此，强化学习问题包含三个主要组成部分：环境、智能体和学习得到的策略。形式上，在每个时间步 t，智能体都会感知其所处环境的状态（$s_t \in S$）。随后，智能体采取一个动作（$a_t \in A$）。采取动作后，智能体会收到奖励（$r_t \in R$），并转移到下一个状态（$s_{t+1} \in S$）。符号 S、 A和 R 分别表示状态空间、动作空间和奖励空间。

我们将（s_t， a_t， r_t， s_{t+1}）这一元组称为智能体与环境交互的记录，其中包含一条观测。智能体与环境交互并收集观测，直到终止状态出现，环境随之结束。回合表示智能体从（初始）状态开始收集经验，直到到达终止状态为止的这段过程。智能体旨在学习策略 $\pi \in \Pi$ （策略空间），使期望累积奖励或回报最大化。

智能体与环境交互并收集奖励时，需要学习如何权衡短期奖励与长期奖励。为便于理解这一概念，计算累积奖励 [13]时会使用折扣因子 (γ)，它是一个介于 0 和 1 之间的超参数。公式 1 展示了如何计算某个状态下策略的价值:

$$V^{\pi}(s) = \mathbb{E}[\, R_t | s_t = s \,], \text{with } R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1)$$

近期，研究人员成功地将深度学习方法融入强化学习，通过 [14] 使用深度神经网络有效地学习策略 [15]–[17]，以解决一些具有挑战性的序贯决策问题。

B. 变异测试与高阶变异

Jia 等人 [4]研究了不同类型的高阶变异，并根据两个主要属性对其进行分类：1）对于给定的一组能够杀死该高阶变异的测试用例$T_H \neq \emptyset$ ，以及给定的一组能够杀死其组成一阶变异的测试用例的并集 $\bigcup_i T_i$，若满足 $T_H \cap \bigcup_i T_i \neq \emptyset$，则该高阶变异为耦合型；2）若满足 $|T_H| < |\bigcup_i T_i|$，则该高阶变异为包含 型，其中 $|\,.\,|$为集合的基数。后者还可进一步细分为强包含型和弱包含型，其中强包含型高阶变异定义为 $T_H \subset \bigcap_i T_i$。从这些定义可以看出，包含型高阶变异尤为重要，因为它们会产生比其组成一阶变异更难检测的变异。因此，聚焦于高阶变异意味着在变异测试中使用更少且更隐蔽的变异体。

C. 深度学习的变异测试

将变异测试应用于深度学习，与将其应用于传统软件工程类似，会引出若干问题，其中最突出的是：测试用例杀死了变异，还是仅仅反映了模型训练的随机性？实际上，给定一个测试用例x、一个神经网络 N和一个变异体 M，$N(x) \neq M(x)$ 并不一定意味着 x 杀死了变异 M，因为变异被杀死可能仅仅是神经网络训练过程中固有的随机性所致。特别是，在相同数据和规范下训练两个神经网络 N_1 和 N_2，并不能保证它们在某个测试用例上的结果一致，因为多种随机性来源可能导致它们彼此产生差异。因此，先前监督学习领域的研究者提出，在使用变异测试 [6], [8] 时，不应只考虑模型的单个实例，而应考虑一组实例。

DeepMutation [6] 提出了对模型（即神经网络）的多个版本的杀死率取平均值这一想法。Jahangirova 等人 [18] 随后，Humbatova 等人 [8] 提出从统计检验的角度，通过比较 n 非变异实例与 n 变异体实例的准确率分布来考量变异测试。具体而言，对于给定的测试集 $\mathcal{T}$ 以及非变异模型的准确率集合 $\mathcal{T}$（A_{N_1}，..... A_{N_n}）和使用给定变异算子生成的变异模型的准确率集合 A_{M_1}（，..... A_{M_n}），他们定义了如下检验函数:

$$MT_{\mathcal{T}} = \begin{cases} 1 & \text{if p-value} < 0.05 \text{ and effectSize} \geq 0.5 \\ 0 & \text{else} \end{cases} \quad (2)$$

其中，p 值 通过使用 广义线性模型 [19] 获得，而 效应量 使用 *Cohen'sd* [20]计算。虽然功效分析并非检验函数的直接组成部分，但他们会利用功效分析排除统计功效过低的检验所对应的变异（阈值为$\beta \geq 0.8$）。

变异测试也已应用于强化学习。文献[12]提出了一种方法，用于评估人工构造的环境基于为强化学习设计的变异算子揭示变异体的能力

。他们将变异体被杀死定义为：对于健康智能体 A 和变异智能体 A_M，在给定环境E 中运行 n 个回合后，两者平均奖励的比值 p_M/p 低于给定阈值 θ。由此可见，与监督学习类似，可能出现这样的情况：(A_1,A_{M_1})能够揭示变异，而 (A_2,A_{M_2})却不能，因为强化学习固有的随机性会显著影响使用相同环境和超参数配置、但随机种子 [21], [22] 不同的训练后智能体的结果。因此，两个用户按照相同的规范训练两个智能体，最终会得到不同的测试结果。

III.（深度）强化学习的变异算子

A. 一阶变异

Lu 等人提出了几种适用于强化学习的变异算子 [12]。然而，这些算子存在若干不足。首先，有些算子无法推广到任意强化学习算法，因此适用范围有限。例如，基于 *epsilon* 参数的变异仅适用于一部分离策略 算法，例如深度Q网络。其次，有些算子（例如在特定层中移除或添加神经元）很可能导致程序崩溃，因为正如 DeepCrime[8] 所指出的，故障本身会引起模型架构的连锁变化。最后，有些变异算子缺乏依据，因为它们并非基于真实故障，这削弱了算子的实用性。例如，在回放缓冲区中打乱回放优先级 并非由任何真实故障所促成。

因此，为了获得相关的一阶变异，我们以 [12] 中定义的算子为基础，并进一步提取了已有的分类体系，这些分类体系报告了影响强化学习或深度学习模型的缺陷 [23] [24]；此外，我们还直接改造了用于监督学习的现有变异算子 [18] [8]。我们得到了一份（非穷尽的）变异算子清单，并根据变异影响的对象将其分为三类（即环境、智能体和策略）。受篇幅限制，本节仅简要介绍这些变异算子。完整说明见我们的复现包 [25] ，其中列出了促成各算子的具体故障依据，并与 [12]中定义的算子进行了比较。

环境级： 环境级变异旨在模拟智能体与环境交互并接收观测时可能发生的故障，即接收到错误的观测。这些故障可能是由故障传感器、环境设计缺陷，甚至恶意攻击造成的 [26], [27]。每种算子还设有应用于某个步骤的概率；概率为 100% 意味着该变异会应用于智能体训练的每一步。

- 奖励噪声 *(RN)*：向智能体原本应接收的真实奖励中添加（高斯）噪声，并将其返回给智能体。
- *Mangled (M)*：破坏所收集经验之间的相关性。该算子会返回一个随机的 s_{t+1} ，并 r_t 这些状态和奖励并非智能体根据其当前状态和所采取的动作应获得的状态和奖励。
- 随机 *(Ra)*：与 Mangled 变异算子类似，*Ra* 变异会向智能体返回一个（s_t, a_t, r'_t, s'_{t+1}）元组。不过，与*M* 算子不同，后者会随机选取 s'_{t+1} 和 r'_t ，且二者之间没有关联；随机算子会向智能体返回从同一经验元组中采样得到、但与 s_t 和 a_t 没有关联的某些 s'_{t+1} 和 r'_t 。
- 重复（*R*）：将先前的观测返回给智能体。如果智能体有两次连续经历，形式分别为（s_t， a_t， r_t， s_{t+1}）和（s_{t+1}， a_{t+1}， r_{t+1}， s_{t+2}），则此算子返回（s_{t+1}， a_{t+1}， r_t， s_{t+1}）。

智能体级： 如 [23] 所示，开发者在实现强化学习算法时面临的许多问题，都是由强化学习概念编码错误导致的。智能体级变异旨在模拟开发者在代码中实现基于强化学习的智能体相关概念时可能犯下的错误。

- 无折扣因子（*NDF*）：从奖励计算中移除折扣因子γ 。
- 缺少终止状态（*MTS*）：移除回合的终止状态（即，到达目标或掉入陷阱.....）。
- 不反转（*NR*）：错误地反转接收到的奖励顺序，因此使智能体学到经历之间的错误关联。
- 缺少状态更新（*MSU*）</style】：智能体执行动作后移除状态更新，这意味着在训练期间，智能体在经验元组 *e.g.*（$s_t, a_{t+1}, r_{t+1}, s_{t+2}$）中始终会看到相同的状态。
- 错误损失函数（ILF）</style】：修改用于训练策略的神经网络中的损失函数。

策略级： 此类别包含影响智能体策略的变异。通常，这些变异应用于神经网络，与此前在监督学习 [8] 中定义的变异类似。

- 策略激活函数变更（PAC）</style】：更改智能体策略网络中使用的默认激活函数。
- 策略优化器变更（POC）</style】：在保持原有学习率不变的情况下，修改算法的默认优化器。

B. 高阶变异

基于我们之前定义的一阶变异，我们也开始评估强化学习中的高阶变异。为识别有趣的高阶变异体，我们采用了与 Jia 等人 [4],在算法1中介绍的类似流程。

从第2行开始，该算法描述了我们在本案例中如何实现 [4] 的启发式方法，以确定哪些一阶变异应纳入高阶变异。我们的目标是确定哪些一阶变异并非平凡，即并非被所有测试环境或一个测试环境都未杀死。为了评估某个变异是否被某个环境杀死，我们对以下对象的奖励进行分布检验：

算法1： 高阶变异生成算法

输入： m一阶变异 $\mathcal{F} = FOM_1, ..., FOM_m$, n健康智能体 $\mathcal{A} = A_1, ..., A_n$, 每个一阶变异对应的相关变异智能体 $\mathcal{A}_{FOM_i} = A_{1,FOM_i}, ..., A_{n,FOM_i}$, 初始环境 E_0, 要修改的参数 $params$

输出： 要纳入考虑的高阶变异的一阶变异集合 $\mathcal{F}^*$

```
1  ℰ ← {E_0, ..., E_p} ← GenerateBoundsEnvironments (𝒜, params, E_0);
2  ℱ* ← ∅;
3  遍历 f ∈ ℱ 执行
4      i ← 0;
5      遍历 e ∈ ℰ 执行
6          如果IsDifferent (𝒜, e, 𝒜_f, e) 那么
7              i++;
8      end
9      如果i ≠ |ℰ| 且 i ≠ 0 那么
10         ℱ* ← ℱ* ∪ f
11 结束
```

在环境e 上评估的健康智能体 $\mathcal{A}$ ，以及在同一环境上评估的变异智能体 $\mathcal{A}_f$ （算法1中的函数 $IsDifferent$）。例如，可以使用第II节提到的分布检验。不过，要应用该启发式方法，需要不同的测试环境。因此，我们需要一种简单而有效地生成更多测试环境的方法。

C. 生成用于变异测试的简单测试环境

生成新测试环境的一种相对直接且可自动化的方法，是修改这些环境的某些属性（即参数）。因此，我们定义测试环境 E_i 取决于其物理参数。例如，CartPole环境[**?**] 由一辆小车和一根通过枢轴与其相连的杆组成。该环境的两个参数是小车和杆的质量，这些参数可以改变，进而影响智能体决策，并由此揭示故障。

要找到多样且非平凡、能够对一阶变异产生不同影响的测试用例并非易事。事实上，由于参数组合数量庞大，而且在每个候选测试环境中评估智能体（无论是否经过变异）行为的计算成本很高，穷举搜索并不实际。与此同时，随机搜索可能会产生许多无关紧要的测试环境，而且同样可能带来高昂的计算成本。因此，需要采用某种形式的启发式方法，哪怕是简单的启发式方法。直观而言，与初始环境接近的环境很可能会针对变异得出相同的决策，这会降低这类环境的相关性。另一方面，若与初始环境相差太远，则可能会意外影响任何智能体的行为，也会降低测试环境的相关性。因此，需要在这两个极端之间找到恰当的平衡。不过，这一方法假定参数空间具有一定的连续性。尽管如此，Biagiola 等人 [28]等采用类似方法的其他研究表明，这种方法是可行的。

因此，我们提出如下方法：仅使用健康智能体即可降低计算成本，并寻找“前沿”环境，即在行为方面与初始环境存在足够差异、但又不会过于不同的环境。一种寻找这一临界点的方法，是在健康智能体于初始环境中的奖励与智能体在生成的候选测试环境中的奖励之间进行二分搜索，并将二者与先前定义的变异杀死定义进行比较（参见第三节-B）。

算法2： 生成边界环境

输入： n 健康智能体 $\mathcal{A} = A_1, ..., A_n$, 初始环境 E_0, 要修改的参数 $params$

输出： 边界环境集合 $\mathcal{E}$

```
1  ℰ ← {};
2  对每个 p ∈ params 执行
       // 上界上的测试环境
3      如果 E_0.p ≠ p.l_upper 那么
4          E_c ← E_0;
5          E_c.p ← p.l_upper;
6          如果 不 IsDifferent (𝒜, E_c, 𝒜, E_0) 那么
7              ℰ ← ℰ ∪ E_c;
8          else
9              E_b ← E_0;
10             当 CheckPrecision (E_c, E_b) 执行
11                 p_m ← (E_b.p + E_c.p)/2;
12                 如果 不 IsDifferent (𝒜, E_c, 𝒜, E_0) 则
13                     E_b.p = p_m;
14                 else
15                     E_c.p = p_m;
16                 end
17             end
18             ℰ ← ℰ ∪ E_b;
19         end
20     else
21         ℰ ← ℰ ∪ E_0;
22     end
23 结束
24 ...;
   /* 下界也类似                                          */
25 ...;
26 对于 i ← 1 到 深度 执行
27     ℰ_m ← {};
28     对于 j ← 1 到 |ℰ|-1 执行
29         如果 ℰ[j] ≠ E_0 且 ℰ[j+1] ≠ E_0 则
30             E_c ← (ℰ[j].p + ℰ[j+1].p)/2;
31             如果非 IsDifferent (𝒜, E_c, 𝒜, E_0) 则
32                 ℰ_⇕ ← ℰ_m ∪ {ℰ[j], E_c, ℰ[j+1]};
33             else
34                 E_b ← E_0;
35                 当 CheckPrecision (E_c, E_b) 执行
36                     p_m ← (E_b.p + E_c.p)/2;
37                     如果不 IsDifferent (𝒜, E_c, 𝒜, E_0) 那么
38                         E_b.p = p_m;
39                     else
40                         E_c.p = p_m;
41                     end
42                 end
43                 ℰ_m ← ℰ_m ∪ {ℰ[j], E_b, ℰ[j+1]};
44             end
45         end
46         ℰ ← ℰ_m;
47 结束
48 如果 E_0 ∉ ℰ 那么
49     ℰ ← ℰ ∪ E_0;
```

这就是算法1第1行中函数 $GenerateBoundsEnvironments$ 的目标，该函数见算法2。

该方法首先沿参数轴搜索测试环境：在初始环境（E_0）的参数与参数 $p.l$ 定义的搜索界限之间，仅针对一个参数进行二分搜索（算法2第2–25行）。在搜索的每一步中，对于健康智能体，都会将其在环境 E_c上获得的奖励分布与在初始环境 E_0上获得的奖励分布进行比较测试（算法2中的函数 $IsDifferent$）。当环境参数的下界/上界之间达到一定精度时，搜索便会停止。接着（第26–46行），算法会遍历预先设定的若干个深度，通过迭代应用相同的方法，在此前计算出的环境之间 搜索测试环境

。请注意，该算法针对二维空间实现，因为我们的实验采用了二维空间，但它可以轻松扩展到 n 维，且不失一般性。因此，对于二维情况，算法从4个点出发，在第一个深度后将得到8个点，第二个深度后则得到16个点。最后，算法1返回测试环境集合。

IV. 实验设计与结果

本节介绍我们的研究问题，并描述所研究环境的实现。此外，我们还将介绍强化学习训练过程、实验设计、所使用的变异杀死定义以及获得的结果。

A. 研究问题

为评估变异测试在强化学习中的有效性，我们提出以下研究问题（RQ）：

研究问题1：现有变异杀死定义应用于强化学习时有哪些局限？**RQ2:** 不同智能体和环境会受到不同变异的何种影响？**研究问题3：**由我们的一阶变异生成的高阶变异是否像传统软件工程中的情况一样具有包含属性？

1) 研究问题*1.* 当前变异杀死定义的局限：正如我们在第II节中详细说明的，变异测试至少可以通过两种方式应用于强化学习。第一种定义由 [12],提出：如果健康智能体在n 个回合中获得的平均奖励与变异智能体的平均奖励之比低于阈值 θ，则认为该变异已被杀死。第二种定义是分布层面的统计检验，例如 DeepCrime[8], 所采用的方法；在强化学习中，该方法可以基于n个回合的奖励，而非测试集上的准确率。然而，第一种定义受到强化学习随机性的限制；第二种定义则是对监督学习中变异测试应用的直接改编，对于强化学习而言可能并不完全适用，因为奖励不一定等同于准确率。**本研究问题旨在评估我们此前定义的变异算子下，这两种变异杀死定义在强化学习中的适用性**。

为了公平比较这两种定义，我们需要修改第一种方法（健康智能体在n 回合中获得的平均奖励与变异智能体平均奖励之比），因为它没有考虑到 N多个智能体。我们将统计在 N 个独立训练的智能体中，该比值低于某个阈值的次数。阈值的选择 θ 会影响发现的变异数量：阈值越高，越有可能发现变异。但与此同时，如果阈值过高 θ，当变异智能体在测试环境中的行为与健康智能体相似时（例如，智能体从变异中恢复），该比值的实用性可能会降低。Lu 等人 [12] 选择的阈值为 $\theta = 0.8$。而我们选择了一个更保守的值 0.9，这会使检验更有可能发现变异。我们将在至少 80% 的 N 比值低于 0.9 时，判定该变异已被杀死；这个比例大致相当于分布级检验所用统计检验的检验效能。第二种方法将对 N 个智能体的奖励进行统计检验，这与 DeepCrime [8]中的原始实现类似。

2）*RQ2*：*FOM* 在生成的测试环境中的行为：本研究问题旨在考察 FOM 在所用变异类型及生成的测试环境方面的行为。测试环境按照第三节中详细介绍的流程生成，深度设为 1。选择这一深度是为了将环境数量控制在较低水平，避免计算开销过大。提高深度会增加生成的环境数量，因此也可能增加相关 FOM 的数量，但代价是处理时间增加。该流程使我们能够通过统计杀死某个 FOM 的测试环境数量，**对一阶变异算子进行更细粒度的分析**。此外，我们还可以分析测试环境的参数，以**评估哪些参数组合更可能触发特定变异**（例如，对于 CartPole，某些变异是否更可能在小车质量降低时被触发）。最后，我们将采用第三节中介绍的方法利用 FOM，**推断出适合用于生成潜在包含型高阶变异的 FOM**。

测试环境将通过修改每个环境的两个参数来生成：对于 CartPole，将修改小车和杆的质量；对于 LunarLander，将修改航天器的重力和侧向发动机功率。有关初始参数以及所用的搜索边界，请参阅我们的复现包 [25]。

3) 研究问题*3*：生成的高阶变异的属性：最后，与 RQ2 类似，在本研究问题中，我们旨在通过重复使用测试环境，**分析先前实验中生成的高阶变异**。我们的目标是了解这些生成的高阶变异具有什么属性，具体依据第 II-B 节中的简要说明。尤其是，**我们旨在探究能否基于所定义的一阶变异和测试环境生成包含型高阶变异**。

使用 RQ2 中选定的一阶变异，我们生成高阶变异，进而用于训练每种环境/算法/变异算子对应的 N 智能体。随后，我们将在先前生成的测试环境中对其进行评估，并将所得结果与 RQ2 中一阶变异所得结果进行比较，以推断其属性。

B. 实现与模型

我们使用 Python 3.8 和 Stable Baselines 3 [29] 框架 1.6.2 作为基础，实现强化学习模型和变异。使用该框架有两个原因：首先，它使我们能够评估一些可能直接影响框架用户的变异，例如为策略网络使用错误的激活函数或优化器。其次，它为实现影响用户可能自定义代码的变异提供了坚实的基础。具体而言，可以通过覆盖基础代码并仅修改

表 I 20 个智能体的平均奖励（括号内为标准差）。

	PPO	A2C	DQN
CartPole	500 (0)	500 (0)	414 (143)
LunarLander	262 (16)	141 (56)	155 (84)

相关部分来引入变异，这样既能降低代码出现非预期错误的风险，也能更好地控制变异实现。

为了测试我们的变异，我们选择了强化学习领域中两个广为人知的环境：*CartPole* 和*LunarLander*。CartPole环境 [?] 由一辆小车组成，杆通过枢轴连接在小车上。智能体的目标是通过让小车向左或向右移动，施加适当大小的水平力，使杆保持稳定并防止其倒下。LunarLander是一个更复杂的环境，它模拟了一艘必须降落在由两面旗帜界定的地表上的宇宙飞船。智能体可以通过调节飞船在三个方向（左、右和下）的推进力来控制它 [30]。

我们使用了三种深度强化学习算法：与上一篇论文类似 [12], ，我们使用深度Q网络（DQN） [15]。此外，我们不再使用相对简单的基础Q学习算法，而是考虑另外两种算法，即优势演员-评论家（A2C）和近端策略优化（PPO），它们都是深度强化学习中的经典算法。这样我们就可以比较 *Off-Policy* 算法（DQN）与*On-Policy* 算法（PPO、A2C）；这是强化学习中的两种主要方法。我们针对每种环境-算法-变异组合，使用了 $N = 20$ 个具有不同随机种子的智能体。表 I 报告了每种算法/环境下，各智能体的平均累积奖励和标准差。我们还提醒读者注意表 II 中使用的变异。

有些变异依赖于某些参数，因此会同时用变异算子标识符和参数来定义。例如，策略激活函数变更需要指定要使用哪种新激活函数，例如随机梯度下降（SGD），因此记为 $PAC\ SGD$。环境级变异根据每个给定步骤中应用变异的概率来定义，因此 $M\ 1.0$ 表示每一步都以 100% 的概率使用变异算子 *Mangled* 。如果不需要参数，则只使用标识符，例如，无反转 只记为 NR。除 *NR*和 *PAC-ReLU* 外，所有变异都用于所有智能体；前者不适用于深度*Q*网络，而后者是深度*Q*网络的默认激活函数。

C. 结果

1) 研究问题*1*：当前变异杀死定义的局限性：表 III 给出了特定算法/环境和杀死定义下的一阶变异结果，其中平均值 表示采用比率平均值的方法，而*R* 表示采用逐奖励统计比较的方法。如图所示，我们在第 II-C 节中简要介绍的平均值 的局限性显而易见：对于多个变异算子，无论环境/算法如何，通过比率评估的智能体都可能

表 II 变异算子汇总

类别	算子	描述
环境级	奖励噪声（RN）	向智能体接收的奖励添加噪声
	Mangled（M）	返回下一个状态和奖励，其中彼此无关
	随机（Ra）	返回彼此相关的下一个状态和奖励彼此相关，但与无关ยเ所采取的动作无关
	重复（R）	返回下一个状态和奖励，来源于先前观测
智能体级	无折扣因子（NDF）	计算累积奖励时不使用折扣因子累积奖励
	不反转（NR）	不反转收到的在计算累积奖励
	缺少状态更新（MSU）	不更新智能体的观测
	缺少终止状态（MTS）	未能保存终止状态观测
	错误的损失函数（ILF）	定义了错误的损失函数（公式错误等）
策略级	策略激活函数变更（PAC）	智能体神经网络使用不同的激活函数网络
	策略优化器变更（POC）	智能体神经网络使用不同的优化器网络

得出截然不同的结果。如果某些变异算子（例如*ILF* 或 *POC-SGD* ）似乎在所有健康/变异智能体对中都被测试环境杀死，那么许多变异被判定为已杀死的智能体对数量相对较少。对于影响相对较小的变异，如果初始化对某个特定健康智能体不利，变异智能体的奖励甚至可能高于健康智能体。因此，训练过程的随机性似乎对所得比率有很大影响，导致变异在多次测试运行中未被杀死。另一方面，使用奖励分布统计检验（*R*）后，除*LunarLander*/深度*Q*网络外，所有情况下被杀死的变异都更多。尤其是，该方法能够杀死一些变异，而使用相同测试环境时，只有少数智能体对将这些变异判定为已杀死AVG 。例如，*CartPole*/优势演员-评论家/*RN-1.0* 中，只有 25% 的比率揭示了该变异。

研究问题1-1 此前介绍的基于健康智能体与变异智能体比例的强化学习变异杀死定义并不充分，因为训练过程的随机性可能导致大量变异未被发现，尤其是那些对训练影响较小、较难发现的变异。基于奖励进行分布层面的统计检验可以改善这一问题。

尽管如此，对于相同的测试环境，按分布进行的*R* 检验仍无法检测出大量变异，即使对于某些变异，按照 *AVG* 定义判定其已被杀死的比例相当高，例如 *LunarLander/A2C/NDF*。在这些情况下，虽然一些变异智能体的奖励可能低于某些健康智能体，但健康智能体和变异智能体的奖励分布可能都未呈现出统计显著的

表 III 一阶变异测试结果。 ✓ 表示变异被杀死，而 ✗ 表示变异未被杀死，或检验的统计功效过低（结论不确定）。使用的杀死判据为 **平均值**：变异体/健康的平均奖励，括号内的值表示低于阈值的比率比例 θ；**R**：基于奖励的统计检验；以及 **DtR**：健康奖励距离统计检验。**"-"**表示不适用该变异。

环境	DRL 算法	杀死判据	变异											
			ILF	M-1.0	R-1.0	Ra-1.0	RN-1.0	NDF	NR	MSU	MTS	PAC-ReLU	PAC-Sigmoid	POC-SGD
CartPole	PPO	AVG	✓ (1.0)	✓ (1.0)	✗ (0.0)	✓ (0.95)	✗ (0.0)	✗ (0.65)	✓ (1.0)	✓ (1.0)	✗ (0.05)	✗ (0.05)	✗ (0.25)	✓ (1.0)
		R	✓	✓	✗	✓	✗	✓	✓	✓	✗	✗	✓	✓
		DtR	✓	✓	✗	✓	✗	✓	✓	✓	✓	✓	✓	✓
	A2C	AVG	✓ (1.0)	✓ (0.95)	✗ (0.15)	✓ (1.0)	✗ (0.25)	✗ (0.35)	✓ (0.9)	✓ (1.0)	✗ (0.05)	✓ (1.0)	✗ (0.2)	✓ (1.0)
		R	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓
		DtR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	DQN	AVG	✓ (1.0)	✓ (1.0)	✓ (0.9)	✓ (1.0)	✗ (0.3)	✓ (1.0)	-	✓ (1.0)	✓ (1.0)	-	✓ (0.8)	✓ (1.0)
		R	✓	✓	✓	✓	✗	✓	-	✓	✓	-	✓	✓
		DtR	✓	✓	✓	✓	✗	✓	-	✓	✓	-	✓	✓
LunarLander	PPO	AVG	✓ (1.0)	✓ (1.0)	✗ (0.2)	✓ (1.0)	✗ (0.05)	✓ (0.95)	✓ (1.0)	✓ (1.0)	✗ (0.05)	✗ (0.05)	✓ (0.95)	✓ (1.0)
		R	✓	✓	✗	✓	✗	✓	✓	✓	✗	✗	✓	✓
		DtR	✓	✓	✗	✓	✗	✓	✓	✓	✓	✗	✓	✓
	A2C	AVG	✓ (1.0)	✓ (1.0)	✗ (0.3)	✓ (1.0)	✗ (0.45)	✗ (0.45)	✓ (0.85)	✓ (1.0)	✗ (0.45)	✗ (0.2)	✗ (0.6)	✓ (1.0)
		R	✓	✓	✗	✓	✗	✗	✓	✓	✗	✓	✗	✓
		DtR	✓	✓	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓
	DQN	AVG	✓ (0.95)	✓ (0.95)	✓ (0.9)	✓ (0.95)	✗ (0.6)	✓ (0.95)	-	✓ (0.95)	✗ (0.7)	-	✓ (0.8)	✓ (0.95)
		R	✓	✓	✓	✓	✗	✓	-	✓	✗	-	✗	✓
		DtR	✓	✓	✓	✓	✗	✓	-	✓	✗	-	✓	✓

差异。当少数智能体最终表现出截然不同的奖励时，尤其可能出现这种情况：这是因为训练未能克服变异的影响，或不利的初始化放大了这种影响，因为初始种子可能会对训练产生重大影响 [21], [22]。删除此类数据点并不理想，因为这会掩盖潜在的有用信息。一种处理这些数据的方法，是利用我们已知健康/变异智能体之间存在差异，同时也知道健康智能体之间 存在差异这一事实。因此，任何潜在的离群值都会导致它们之间出现较大差异。可以通过计算健康智能体分布样本之间的奖励的海林格距离[31] （内部 距离），以及健康/变异智能体分布样本之间的相同距离（外部 距离），来估算这种差异。海林格距离是一种取值范围为 0 到 1 的度量，其中 0 表示两个分布相同，因此可以将其视为衡量分布相似性的指标。对于离散情形，两个分布 P 和 Q 的定义如下：

$$H(P,Q) = \frac{1}{\sqrt{2}}||P-Q||_2 \tag{3}$$

其中 $||.||_2$ 是 L^2 范数。

通过从两个智能体的奖励分布中采样并重复计算，可以获得间/内 距离的分布，并使用定义*R*中采用的相同统计检验。实际中，我们将从 20 个智能体中抽取 10 个样本来计算距离。结果列于表 III 中*DtR* 对应的行。

与仅使用健康/变异智能体的奖励相比，采用这种方法效果更好。具体而言，由于*DtR*能够考虑此前可能存在的离群值问题，因此它可以让同一测试环境（例如 *LunarLander/PPO/MTS*）杀死一些平均值 得分相对较低的变异体。尽管如此，该方法仍不完美，例如 *LunarLander/DQN/RN* 等变异最终并未被杀死，而平均值则将相当数量的智能体对判定为变异体。在这些情况下，健康智能体之间的差异可能远大于健康智能体与变异智能体之间的差异，这就是*DtR* 无法检测出变异的原因。不过，这凸显了一项重要观察：在强化学习中，变异杀死定义的设计是决定变异是否被杀死的关键步骤。

研究问题1-2 除了选择考虑哪些变异算子之外，谨慎选择将变异测试应用于强化学习时如何 使用它，也是一个重要参数。例如，与直接使用奖励相比，使用海林格距离衡量与健康奖励的差异，能在相同测试环境下杀死更多变异。因此，研究强化学习中不同变异杀死定义的设计至关重要。

2) 研究问题*2*。一阶变异在生成的测试环境中的行为：按照第三节-C的流程，我们为每种算法/环境生成了若干测试环境，并进行了多项观察。结果列于表IV。对于给定的环境/算法，标记为绿色的变异表示该情况下的变异并非显而易见，因此在需要生成高阶变异体时值得关注。请注意，从这里开始，我们只关注分布级测试，因为研究问题1指出了[12]中变异杀死定义的缺点。

我们首先观察到，有五种变异会被所有生成的测试环境检测出来，分别是*ILF*、*MSU*、*NR*、*POC-SGD*、*M-1.0* 和 *Ra-1.0*。这些变异的影响太大，不可能不被检测出来；表 III 也凸显了这一点，因为使用初始测试环境和比率法（平均值）时，这些变异得到的分数为 > 0.85 。在这些情况下，变异的高影响掩盖了随机性，所有训练后的智能体表现都相似。因此，生成多个测试环境让我们得以看出

这些变异可能并不太值得关注。事实上，它们相当容易检测，因此很可能可以忽略。

其次，我们发现，与在*LunarLander*上生成的测试环境相比，在*Cart-Pole* 上生成的测试环境更有可能检测出变异，其中大多数变异至少会被一半的测试环境杀死。尽管生成的测试环境数量可能会产生影响，但这很可能是因为*LunarLander* 比*CartPole*更复杂，因此，随着环境复杂度增加，变异可能会更难检测。例如，*CartPole*似乎对*MTS* 或*PAC* 变异不太敏感；*i.e.*，移除回合的终止状态或更改策略网络的激活函数，似乎不会对在*CartPole* 上训练的智能体产生太大影响，这与*LunarLander*的情况相反。

RQ2-1 与仅使用初始测试环境相比，生成额外的测试环境使我们能够根据杀死某个变异的环境数量，大致评估哪些变异可能无关紧要，哪些更值得关注。我们还发现，环境似乎越复杂，越有可能检测不到变异。

第二步，我们可以进一步分析杀死某个变异的测试环境，而不只是统计这类测试环境的数量。具体来说，可以检查究竟是哪个测试环境杀死了该变异。由于我们是通过修改初始环境的参数来生成测试环境的，因此这有助于我们了解变异可能更容易对哪些参数敏感。通过这种方式，我们可以确定哪些测试环境有助于识别某些潜在故障，进而形成某种故障检测方法。受篇幅所限，我们将使用*R* 给出一个示例，完整的原始结果请参阅我们的实现代码库 [25] 。对于*LunarLander*/近端策略优化/*PAC ReLU*，如图 1 所示，与初始环境相比，重力（绝对值）较低的测试环境会杀死该变异，而重力（绝对值）较高的测试环境则不会。因此，该变异似乎在某种程度上受重力参数的影响。当重力接近初始环境中的值时，较高的发动机功率对于杀死该变异至关重要。

测试环境以橙色/蓝色圆点表示，具体颜色取决于它们是否杀死变异。可以看到，按照

前文的描述。具体来说，我们可以看到，测试环境 $P1$ 和 $P2$ 似乎都接近该变异的某些边界，因为它们在参数空间中彼此接近，却对该变异得出了相反的判断。请注意，红色边界是任意设定的，因为我们无法获得精确边界；正如第三节所述，寻找精确边界的计算开销可能过高。因此，$P3$ 与初始环境之间可能还存在其他环境，仍能触发该变异。我们只知道，对于环境 $P3$ ，健康智能体的奖励分布与初始环境中观察到的分布已接近出现差异的临界点

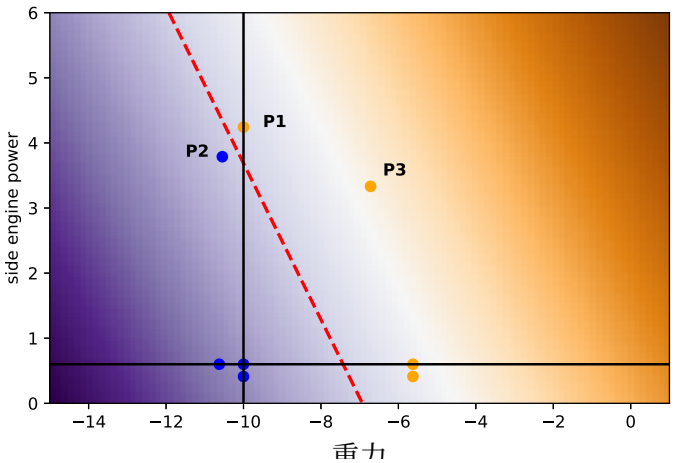

图1. 为 *LunarLander*/近端策略优化/*PAC ReLU* 生成的测试环境，以及根据它们是否杀死变异对其进行区分的一种可能方法。 橙色 圆点表示杀死变异，而 蓝色 圆点则表示没有杀死变异。原点位于初始环境的中心。

，但它没有提供有关变异智能体分布的信息。虽然对于所有变异，尤其是在测试环境数量较少时，可能无法得出有意义的结论，但这仍然表明，以这种方式生成测试环境，也能让我们探索环境参数与其对变异的影响之间可能存在的联系。

RQ2-2 通过针对给定变异，映射哪些生成的测试环境会杀死它、哪些不会，我们可以分析测试环境中的哪些参数会影响是否杀死该变异。这概述了某种形式的故障检测方法。

3) 研究问题*3*。生成的高阶变异的性质： 按照RQ2的做法，我们针对每种环境/算法/杀死定义，收集了非平凡一阶变异（即，既非被所有测试环境杀死，也非完全未被杀死），见表IV。由于我们只研究二阶高阶变异，因此生成高阶变异至少需要两个一阶变异。随后，我们按照与一阶变异相同的方式，基于收集到的高阶变异训练新的变异智能体。最后，根据变异杀死定义（*R*或 *DtR*），使用此前生成的测试环境评估变异智能体，并依据我们在第II-B节中介绍的分类方法分析高阶变异的类型。结果见表 V。

正如我们所见，按照 RQ2 中提到的流程，我们最终得到的非平凡一阶变异数量并不多，因此生成的高阶变异池相对较小，甚至有些配置未能生成任何高阶变异。尽管如此，在使用*R* 变异杀死定义时，我们成功获得了 12 个高阶变异；使用*DtR* 时则获得了 23 个，但仅限于*LunarLander*。其中，采用*R*方法时，非包含型（*NS*）占高阶变异的 50%；而采用*DtR*时，这一比例仅为 30%。两种方法之间的差异可能是由*DtR*更高的敏感性所致。其余包含型高阶变异大多属于弱包含耦合型 （WSC）

表IV 杀死每个一阶变异的测试环境数量。绿色单元格中的一阶变异将用于生成高阶变异。

环境	DRL 算法	杀死判据	变异											
			ILF	M-1.0	R-1.0	Ra-1.0	RN-1.0	NDF	NR	MSU	MTS	PAC-ReLU	PAC-Sigmoid	POC-SGD
CartPole	PPO	R	4/4	4/4	0/4	4/4	3/4	4/4	4/4	4/4	3/4	3/4	4/4	4/4
		DtR	4/4	4/4	3/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
	A2C	R	4/4	4/4	0/4	4/4	4/4	3/4	4/4	4/4	3/4	4/4	2/4	4/4
		DtR	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
	DQN	R	4/4	4/4	4/4	4/4	0/4	4/4	-	4/4	4/4	-	4/4	4/4
		DtR	4/4	4/4	4/4	4/4	3/4	4/4	-	4/4	4/4	-	4/4	4/4
LunarLander	PPO	R	8/8	8/8	0/8	8/8	1/8	8/8	8/8	8/8	0/8	4/8	8/8	8/8
		DtR	6/8	6/8	2/8	6/8	3/8	6/8	6/6	6/6	3/6	3/6	6/6	6/6
	A2C	R	9/9	9/9	0/9	9/9	0/9	6/9	9/9	9/9	2/9	3/9	5/9	9/9
		DtR	9/9	9/9	2/9	9/9	4/9	7/9	9/9	9/9	4/9	7/9	6/9	9/9
	DQN	R	9/9	9/9	8/9	9/9	0/9	9/9	-	9/9	0/9	-	0/9	9/9
		DtR	9/9	9/9	9/9	9/9	4/9	9/9	-	9/9	6/9	-	4/9	9/9

表 V 使用 RQ2 非平凡一阶变异生成的高阶变异类型。若未生成高阶变异，则使用"-"。非包含型：非包含，弱包含耦合型：弱包含耦合型，弱包含解耦型：弱包含解耦型，强包含耦合型：强包含耦合型。

环境	DRL 算法	杀死判据	高阶变异体类型				
			HOM	NS	WSC	WSD	SSC
CartPole	PPO	R	3	1	0	0	2
		DtR	-	-	-	-	-
	A2C	R	3	2	1	0	0
		DtR	-	-	-	-	-
	DQN	R	-	-	-	-	-
		DtR	-	-	-	-	-
LunarLander	PPO	R	1	1	0	0	0
		DtR	6	1	5	0	0
	A2C	R	5	2	3	0	0
		DtR	14	6	8	0	0
	DQN	R	-	-	-	-	-
		DtR	3	0	3	0	0

类型，并且通常占每种配置所生成高阶变异的一半以上，这与 Jia 等人 [4] 在一些传统软件程序中得到的高阶变异结果相似。有趣的是，我们发现了 2 个符合强包含耦合型 （SSC）类型的高阶变异，也就是说，能够杀死这些高阶变异的测试环境也会杀死其组成一阶变异的高阶变异。除此之外，我们没有发现其他*SSC* ，也没有发现*WSD* 。即使在 Jia 等人的案例中，*SSC* 也很少见（占包含型高阶变异的 < 1%），因此，考虑到我们找到的高阶变异和测试环境数量有限，没有发现其他此类变异也不足为奇。尽管如此，我们证明了包含型高阶变异（即使是弱包含 型）这种更复杂、更微妙的变异体也能够生成，并且占生成的高阶变异的一半以上。

研究问题3 由非平凡一阶变异生成的高阶变异虽然数量不多，但大多数都属于包含型高阶变异，正如我们在 II-B节中指出的，这实际上使它们成为更值得研究的案例。虽然像*SSC* 这样更微妙的包含型高阶变异并不常见，但能够生成其中一些，表明在强化学习中也可以实现这一属性。

V. 有效性威胁

构念效度： 变异测试中的设计选择可能会影响我们的结果。我们的主要目标并非为深度学习设计新的变异杀死定义，而是调整并评估现有方法的表现，因此，本文使用的所有变异杀死定义均基于现有方法。此外，尽管所用超参数可能会影响对变异体是否被杀死的判定（p 值、阈值 θ.....），但我们倾向于沿用每种杀死定义在其原始实现中给出的超参数，并将研究超参数影响的工作留待未来开展。我们搜索参数空间以生成测试环境的方式也可能影响了结果。尽管如此，本研究的目标只是提出一种简单有效、可自动化的方法，为后续工作提供基础启发式方法。我们方法中的其他部分，例如对高阶变异属性的定义，均以科学文献为依据，并沿用其原始定义。

内部效度： 所选择的变异算子可能会影响我们的结果。虽然我们无法做到面面俱到，但我们确保基于已有故障或分类体系创建算子，并在对代码的影响方面提供足够的多样性（基于智能体、基于环境、基于策略）。对于基于环境 的算子，尽管应用概率为 1.0 可能无法真实模拟传感器默认值，但当概率设置为< 1.0时，为避免对变异会影响哪个步骤产生潜在偏差，这种做法是必要的。为生成测试环境而修改的环境参数也可能影响结果。不过，我们的目标是展示可以自动生成简单有效的测试环境，并借此分析强化学习中的变异测试。因此，参数本身的选择并没有那么重要。

外部效度： 环境类型和算法的选择可能影响我们实验结果的泛化能力。我们确保选择了两种环境以及三种通常用作强化学习基准测试的算法。

可靠性效度： 实现变异可能会导致一些意外故障。我们使用 Stable-Baselines

框架作为基础来实现我们的变异，以降低风险。我们还提供了我们的实现和相关工件[25]。

VI. 相关工作

Gur 等人 [32] 提出了一种智能体，该智能体会根据学习智能体的技能水平生成环境。他们的方法能够以更有系统性的方式训练智能体，并增强其鲁棒性和泛化能力。尽管这一方法颇具价值，但它将环境生成器与智能体联合训练，这超出了变异测试的范畴。为解决过拟合问题，Cobbe 等人 [33] 提出了 Procgen 基准。该基准包含 16 个程序生成的环境，研究人员可用其测试智能体的泛化能力。虽然他们的方法能够用于测试智能体，但变异测试主要关注的是评估测试集质量。尽管如此，未来不妨在这些环境中评估 *RLMutation*，因为这些环境旨在考验算法的泛化能力，因此可能会引发不同的变异行为。另一方面，基于元启发式方法的算法也可能成为改进相关测试环境生成的一条途径 [34]。最后，Biagiola 等人 [28] 致力于通过结合二分／指数搜索来生成测试环境，从持续学习的角度描绘智能体的适应性与防回归性能。在我们的工作中，我们同样通过调整环境参数来生成测试环境，但不测试适应性，也不通过持续学习重新训练智能体。我们的做法与其说如此，不如说是同样在初始和生成的测试环境中测试健康智能体的防回归能力。尽管如此，他们的方法仍可用于改进我们的测试生成部分。

如引言和第II节所述，一些工作将变异测试应用于监督学习 [6]–[8]。虽然我们借鉴了其中一些变异算子和设计选择，但我们专门针对强化学习定制了自己的方法。Lu 等人 [12]也在强化学习中研究了变异测试，并探讨了精心设计的环境如何用于揭示某种特定变异；这更接近故障检测方法。相反，我们着重以自动化方式生成简单而有效的测试环境，用于揭示更复杂的故障，即高阶变异（HOM）。我们还表明，由于强化学习的随机性，他们在变异测试中的设计选择可能会成为限制因素。Shen 等人 [35] 利用监督学习模型的决策边界，寻找更有可能触发变异的测试集子集。从某种意义上说，我们的方法与他们类似，但视角来自强化学习：我们旨在寻找处于某些边界上的测试环境。然而，与他们不同的是，我们没有可供使用的实际测试集，因此必须生成测试环境。此外，我们并不假设位于该边界上的测试环境更有可能揭示变异。从启发式角度而言，我们只是假设这些环境是值得关注的潜在点；在生成测试环境数量受限时，它们优于任意随机点。并进行测试。最后，Tambon 等人 [36] 深入研究了在监督学习中采用 DeepCrime 方法进行变异测试时，深度学习模型实例的选择对结果的影响。他们表明，这种选择会影响变异测试的结果，而贝叶斯方法可以缓解这一问题。在我们的方法中，我们没有考虑这种影响，而是专注于现有的杀死定义和测试环境生成，因为我们的主要目标是分析强化学习中的一阶变异及后续的高阶变异。尽管如此，验证强化学习中是否也存在类似现象，将会很有意思。

七、结论

本文提出了 *RLMutation*，这是一个用于强化学习变异测试的框架。我们根据深度强化学习智能体在设计和训练过程中可能出现的真实故障，定义了三类不同的一阶变异。我们基于此前将变异测试应用于深度学习的框架，对不同的变异杀死定义方案进行了比较。我们还评估了高阶变异（由一阶变异组合而成），发现它们可能更复杂、更难杀死，因此值得研究其中有趣的边界故障。生成高阶变异时所用的一阶变异，是根据其与所生成参数化测试环境的相关性选取的。我们在两种基准测试环境（LunarLander 和 CartPole）中，对一组先进的强化学习算法（深度Q网络、优势演员-评论家和近端策略优化）测试了我们的方法。研究结果表明，变异杀死定义的选择对于杀死变异至关重要（比率与奖励分布）。我们证明，在修改后的环境中测试智能体，可以检测出在其训练环境中测试时无法发现的非平凡一阶变异。最后，我们表明，由非平凡一阶变异生成的高阶变异大多具有有趣的包含性：它们比其组成的一阶变异更难杀死，因此从测试角度来看更值得关注。

总体而言，本研究表明，尽管将变异测试应用于强化学习面临诸多挑战，包括变异杀死定义的设计、相关测试环境的生成以及高阶变异，但这是改进基于强化学习的软件系统测试的一条颇具意义的途径。因此，我们认为，未来对此主题的研究应聚焦于这些问题，以进一步完善应用于强化学习的变异测试。

致谢

本研究获得以下机构资助：魁北克研究基金（FRQ）、加拿大高级研究所（CIFAR），以及由加拿大国家科学与工程研究委员会（NSERC）和魁北克航空航天研究与创新联盟（CRIAQ）共同资助的 DEEL 项目 CRDPJ 537462-18；该联盟的工业合作伙伴包括泰雷兹加拿大公司、贝尔德事隆加拿大有限公司、CAE 公司和庞巴迪公司。

参考文献

[1] Y. Jia 和 M. Harman，“变异测试发展的分析与综述”，IEEE软件工程汇刊，第 37 卷，第 5 期，第 649–678 页，2011 年。[2] M. Papadakis、M. Kintis、J. Zhang、Y. Jia、Y. Le Traon 和 M. Harman，“变异测试进展：分析与综述”，载于《计算机进展》，第 112 卷，第 275–378 页，爱思唯尔，2019 年。[3] A. J. Offutt，“软件测试耦合效应研究”，ACM软件工程方法论汇刊，第 1 卷，第 5–20 页，1992 年 1 月。[4] Y. Jia 和 M. Harman，“高阶变异测试”，《信息与软件技术》，第 51 卷，第 10 期，第 1379–1393 页，2009 年。源代码分析与处理，SCAM 2008。[5] J. M. Zhang、M. Harman、L. Ma 和 Y. Liu，“机器学习测试：综述、领域概况与发展前景”，IEEE软件工程汇刊，第 48 卷，第 1 期，第 1–36 页，2022 年。[6] L. Ma、F. Zhang、J. Sun、M. Xue、B. Li、F. Juefei-Xu、C. Xie、L. Li、Y. Liu、J. Zhao 等，“Deepmutation：深度学习系统的变异测试”，载于《2018年IEEE第29届国际软件可靠性工程研讨会（ISSRE）》，第 100–111 页，IEEE，2018 年。[7] Q. Hu、L. Ma、X. Xie、B. Yu、Y. Liu 和 J. Zhao，“Deepmutation++：深度学习系统的变异测试框架”，载于《2019年第34届IEEE/ACM国际自动化软件工程会议（ASE）》，第 1158–1161 页，IEEE，2019 年。[8] N. Humbatova、G. Jahangirova 和 P. Tonella，“Deepcrime：基于真实故障的深度学习系统变异测试”，载于《第30届ACM SIGSOFT国际软件测试与分析研讨会论文集》，第 67–78 页，2021 年。[9] N. C. Luong、D. T. Hoang、S. Gong、D. Niyato、P. Wang、Y.-C. Liang 和 D. I. Kim，“深度强化学习在通信与网络中的应用：综述”，IEEE通信调查与教程，第 21 卷，第 4 期，第 3133–3174 页，2019 年。[10] A. S. Polydoros 和 L. Nalpantidis，“基于模型的强化学习综述：在机器人学中的应用”，《智能与机器人系统期刊》，第 86 卷，第 2 期，第 153–173 页，2017 年。[11] V. Franc¸ois-Lavet、P. Henderson、R. Islam、M. G. Bellemare 和 J. Pineau，“深度强化学习导论”，《机器学习基础与趋势》，第 11 卷，354，2018 3 4，219 第 - 页 –。[12] Y. Lu、W. Sun 和 M. Sun，“迈向强化学习系统的变异测试”，《系统架构期刊》，第 131 卷，102701，2022 年，第 页。[13] R. S. Sutton 和 A. G. Barto，《强化学习：导论》。麻省理工学院出版社，2018 年。[14] I. Goodfellow、Y. Bengio 和 A. Courville，《深度学习》。麻省理工学院出版社，2016 年。http://www.deeplearningbook.org。[15] V. Mnih、K. Kavukcuoglu、D. Silver、A. A. Rusu、J. Veness、M. G. Bellemare、A. Graves、M. Riedmiller、A. K. Fidjeland、G. Ostrovski 等，“通过深度强化学习实现人类水平的控制”，《自然》，第 518 卷，第 7540 期，第 529–533 页，2015 年。[16] D. Gandhi、L. Pinto 和 A. Gupta，“在坠机中学会飞行”，载于《2017年IEEE/RSJ智能机器人与系统国际会议（IROS）》，第 3948–3955 页，IEEE，2017 年。[17] M. Morav˘c´ık、M. Schmid、N. Burch、V. Lis`y、D. Morrill、N. Bard、T. Davis、K. Waugh、M. Johanson 和 M. Bowling，“Deepstack：单挑无限注扑克中的专家级人工智能”，《科学》，第 356 卷，第 6337 期，第 508–513 页，2017 年。[18] G. Jahangirova 和 P. Tonella，“深度学习系统变异算子的实证评估”，载于《2020年IEEE第13届国际软件测试、验证与确认会议（*ICST*），第74–84页，2020年。[19] J. A. Nelder 和 R. W. M. Wedderburn，“广义线性模型”，英国皇家统计学会会刊：*A*辑（综合），第135卷，第3期，第370–384页，1972年。[20] K. Kelley 和 K. J. Preacher，“论效应量”，心理学方法，第17卷，第2期，第137–152页，2012年。[21] P. Henderson、R. Islam、P. Bachman、J. Pineau、D. Precup 和 D. Meger，“值得关注的深度强化学习”，载于第三十二届*AAAI*人工智能会议、第三十届人工智能创新应用会议及第八届*AAAI*人工智能教育进展研讨会论文集，AAAI’18/IAAI’18/EAAI’18，AAAI出版社，2018年。[22] R. Agarwal、M. Schwarzer、P. S. Castro、A. C. Courville 和 M. G. Bellemare，“处于统计悬崖边缘的深度强化学习”，载于*NeurIPS*，2021年。[23] A. Nikanjam、M. M. Morovati、F. Khomh 和 H. Ben Braiek，“深度强化学习程序中的故障：分类体系与检测方法”，自动化软件工程，第29卷，32，2022 1，1第。页–。[24] N. Humbatova、G. Jahangirova、G. Bavota、V. Riccio、A. Stocco 和 P. Tonella，“深度学习系统中的真实故障分类体系”，载于第*42*届*ACM/IEEE*国际软件工程会议论文集，第1110–1121页，2020年。[25] “复现包。”https://github.com/FlowSs/RLMutation，2022年。[26] T. Everitt、V. Krakovna、L. Orseau、M. Hutter 和 S. Legg，“奖励通道受损的强化学习”，2017年。[27] J. Romoff、P. Henderson、A. Piche、V. Francois-Lavet 和 J. Pineau，“深度强化学习中的奖励估计与方差缩减”，载于第*2*届机器人学习会议论文集 （A. Billard、A. Dragan、J. Peters 和 J. Morimoto 编），机器学习研究会议录第87卷，第674–699页，PMLR，2018年10月29–31日。[28] M. Biagiola 和 P. Tonella，“测试基于强化学习的系统的可塑性”，*ACM*软件工程与方法学汇刊，第31卷，2022年7月。[29] A. Raffin、A. Hill、A. Gleave、A. Kanervisto、M. Ernestus 和 N. Dormann，“Stable-baselines3：可靠的强化学习实现”，机器学习研究期刊，第22卷，8，2021 268，1第。页–。[30] “月球着陆器。”https://www.gymlibrary.dev/environments/box2d/lunar lander/，2022年。[31] H. Cramer，统计学数学方法。普林斯顿大学出版社，1946年。[32] I. Gur、N. Jaques、Y. Miao、J. Choi、M. Tiwari、H. Lee 和 A. Faust，“零样本组合式强化学习的环境生成”，神经信息处理系统进展，第34卷，第4157–4169页，2021年。[33] K. Cobbe、C. Hesse、J. Hilton 和 J. Schulman，“利用程序化生成对强化学习进行基准测试”，载于国际机器学习会议，第2048–2056页，PMLR，2020年。[34] D. Humeniuk、F. Khomh 和 G. Antoniol，“用于自动生成信息物理系统测试环境的基于搜索的框架”，信息与软件技术，第149卷，106936，2022年第。页。[35] W. Shen、Y. Li、Y. Han、L. Chen、D. Wu、Y. Zhou 和 B. Xu，“通过边界采样提升深度学习模型的变异测试”，信息与软件技术，第130卷，第106413页，2021年。[36] F. Tambon、F. Khomh 和 G. Antoniol，“深度神经网络变异测试的概率框架”，信息与软件技术，第155卷，第107129页，2023年。