Machine Learning Testing: Survey Landscapes and Horizon

Chen Gong

02 June 2021

目录

1 Core idea

机器学习测试是一个非常有意思的概念，算法的鲁棒性和稳定性等等，对于未来机器学习算法在工业界应用非常重要，不可忽视。而随着机器学习的广泛应用，人们将越来越关注机器学习算法的可信度，算法是否是公平的，是否是鲁棒的，有效的，它是否能很好的保护我们的隐私。**软件测试的概念，旨在检测任何现有行为和所需行为之间存在差异的活动**。所以测试是一种有效的暴露问题的方法，它有助于我们提高机器学习系统的可信度。

本文是一篇有关于机器学习测试的综述，于 2020 年发表与软件工程领域顶级期刊 **IEEE Transactions on Software Engineering**。本文对 machine learning testing 做了比较全面的调研，其内容：涵盖测试属性 (如正确性、鲁棒性、公平性)、测试组件 (如数据、学习程序、框架)、测试工作流 (如测试生成、测试评估)、应用场景 (如自动驾驶、机器翻译) 等等。分析了机器学习测试，数据集发展趋势、研究趋势和研究重点，提出了机器学习测试的研究挑战和发展方向。

2 Introduction

机器学习系统的统计学特性，及其自主决策的能力为软件测试提出了很多非常具有挑战性的研究问题。与传统的软件系统 testing(相对更确定性，较少面向统计) 相比，在本质和构造上有着根本的不同，这给机器学习测试带来了挑战。首先要介绍的是 Machine Learning 中的困难。

2.1 Machine Learning Testing 中的困难

1. **模型的多变性**。因为机器学习系统本质上是一种数据驱动的编程范式，决策逻辑是通过不断的学习数据来进行的。那么模型的行为可能会随着时间的改变而不断的改变，当模型遇到了新的数据的时候，模型会改变自己以适应新的数据。传统的软件测试其核心底层行为通常不会像机器学习系统那样随着新数据的出现而改变。

2. **Oracle 问题**。学习系统很难被测试，因为他们实际上被设计是用于解决一个不知道答案的问题。我们能够知道正确答案，就不需要编写这样的程序了。

3. **Emergent Properties**。此性质描述的是对于机器学习系统通常需要整体的去考虑。精度可能受到数据集，学习程序，学习框架，等等非常多的不同的组件的影响。当时测试变得非常的困难。需要把整个测试元件分成非常多的小块去单独测试，这个分解的过程也是非常的麻烦的。

4. **对错误的判断不准确**。错误可能会被放大或抑制，从而让测试人员忽略某些故障。这个问题在传统的软件测试中也会遇到。机器学习的测试中这个问题将变得影响非常的大。为它们产生于机器学习方法的本质，并从根本上影响所有行为，而不是传统数据和控制流的副作用。**个人理解，机器学习中错误本就是一件很正常的事情，所以我们很容易忽视一些错误**。

由于这些原因，之前有了文献中认为机器学习是**不可测试**的。随着研究的不断深入，这些问题逐渐得到了好转，于是从 2016 年开始，新的软件测试领域出现了：机器学习测试。

将从 4 个方面全面的分析机器学习测试：1. 测试属性 (如正确性、鲁棒性、公平性)；2. 测试组件 (如数据、学习程序、框架)；3. 测试工作流 (如测试生成、测试评估)；4. 应用场景 (如自动驾驶、机器翻译)。本文的贡献如下所示，

1. Definition: Machine Learning Test 的定义，包括概念，测试工作流，测试属性，和机器学习测试的相关原件。
2. Survey: 该论文提供了 144 篇机器学习测试论文的综合调查，涉及软件工程、人工智能、系统和网络以及数据挖掘等不同领域。
3. Analyses: 本文分析和报告了机器学习测试文献的研究分布、数据集和趋势。研究成果的分布明显不平衡，在作者收集的 144 篇论文中，大约 120 篇涉及监督学习测试，3 篇涉及无监督学习测试，只有 1 篇论文测试强化学习。此外，大多数论文 (93 篇) 关注正确性和健壮性，但只有少数论文测试可解释性、隐私性或效率。
4. Horizons：本文中定义了挑战，开放性问题，还有未来可能的研究方向。

3 Perliminaries of Machine Learning

首先介绍的是 machine learning 的一些基本概念。这里只描述几个不太好理解，容易混淆的概念。
Learning program: 写出来的程序；
Framework: 开发平台，如 Pytorch，TensorFlow 等；
Instance: 记录一个对象的信息的一段数据。
Test error: 实际条件与预测条件的差值比。
Generalisation error: 任何有效数据的实际情况与预测情况之间的期望差值比。

4 Machine Learning Testing

本小节描述的是如何定义和分析 ML testing，包括描述了测试工作流（怎么测试），测试属性（测试得到什么）和测试组件（在哪里测试）。

4.1 Definition

软件 bug 是指计算机程序中存在的缺陷，导致存在的条件和要求的条件不一致。本文中，术语“bug” 指的是 ML 系统的现有行为和所需行为之间的差异。
(Definition 1) ML Bug: ML bug 是指机器学习中导致现有条件和所需条件不一致的任何缺陷。而软件测试的目的就是发现 ML Bug。
(Definition 2) ML Testing: ML Testing 是指任何旨在发现机器学习错误的活动。

值得注意的是，在 ML testing 中，我们不仅仅要测试程序中所包含的 bug，同时也要考虑数据中不合理的地方。这导致 ML testing 比传统的软件测试形式更加的丰富，更加的多样化。当我们试图检测数据中的 bug 时，甚至可以使用训练程序作为测试输入来检查数据所需的一些属性。

4.2 ML Testing Workflow

ML testing 工作流是关于如何用不同的测试操作进行机器学习测试。在本节中，我们首先简要介绍了 ML 测试在构建 ML 模型中的作用，然后介绍了 ML 测试中的关键步骤和活动。我们将在 Section 5 中详细介绍目前关于每个步骤的研究。

4.2.1 ML testing 在 ML 发展中的作用

下图中展示了部署包含 ML testing 的机器学习系统的生命周期。一开始，基于历史数据生成原型模型；在在线部署模型之前，需要进行离线测试，例如交叉验证，以确保模型满足所需的条件。部署之后，模型会进行工作，产生新的数据，这些数据可以通过在线测试进行分析，以评估模型如何与用户行为交互。

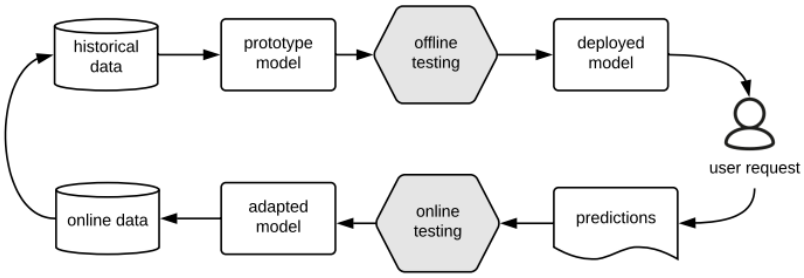

图 1: ML testing 在 ML 系统发展中的作用图

其中，有几个原因使得在线测试变得至关重要。首先，离线测试通常依赖于测试数据，而测试数据通常不能完全代表未来数据。其次，离线测试无法测试在实际应用场景中可能出现问题的某些情况，如数据丢失和调用延迟。此外，离线测试无法访问一些业务指标，如打开率、阅读时间和点击率。

下面，我们提出了一个基于经典软件测试工作流的 ML testing 工作流。下图中显示了工作流，包括离线测试和在线测试。

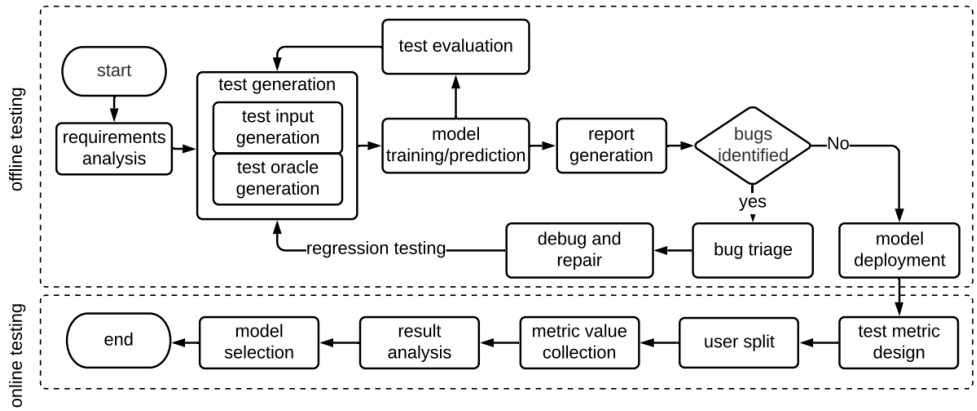

图 2: 一个基于经典软件测试工作流的 ML testing 工作流

4.2.2 Offline Testing

一开始，开发人员需要进行需求分析，定义用户对被测机器学习系统的期望，想要这个系统干什么。在需求分析中，对机器学习系统的性能指标进行了分析，并规划了整个测试过程。然后测试输入，要么从收集的数据中采样，要么基于某种特定的目的生成测试样例。然后识别或生成测试 oracle。当测试准备好后，需要为开发人员执行测试以收集数据。测试执行过程包括使用测试构建模型 (当测试

是训练数据时) 或针对测试运行构建模型 (当测试是测试数据时)，以及检查是否违反了测试 oracle。在测试执行过程之后，开发人员可以使用评估指标来检查测试的质量，例如，测试发现 ML 问题的能力。

测试执行结果会生成 bug 报告，以帮助开发人员复制、定位和解决 bug。识别出来的错误将被贴上不同严重程度的标签，并分配给不同的开发人员。一旦错误被调试和修复，就要进行回归测试，以确保修复解决了报告的问题，而不会带来新的问题。如果没有发现 bug，则离线测试过程结束，并部署模型。

4.2.3 Online Testing

离线测试使用历史数据来测试模型，而不是在真实的应用程序环境中。缺少用户行为的数据收集过程。在线测试弥补了离线测试的不足，旨在模型在线部署后检测 bug。针对不同的目的，有不同的在线测试方法。例如，运行时监视不断检查运行中的 ML 系统是否满足需求。另一个常用的场景是监视用户响应，并根据用户响应确定在某些应用程序中，新模型是否优于旧模型。A/B 测试就是这种在线测试的典型类型。它将客户分开来比较系统的两个版本孰优孰劣。在 ML 系统上进行 A/B 测试时，用户将被随机分为两组，分别使用新的和旧的 ML 模型。

MAB (Multi-Armed Bandit，多臂赌博机) 是的另一种在线测试方法。它首先进行短时间的 A/B 测试，找出最佳模型，然后在选定的模型上投入更多资源。

4.3 ML Testing 组件

在部署 ML 学习模型的时候，机器学习软件开发者需要收集数据标记数据，设计学习算法，并且使用这种学习算法。所以在设计机器学习算法的过程中，需要和各种各样的元件进行交互，比如数据，学习程序以及学习框架，每个组件都有可能会产生 bug。下图中显示了构建 ML 模型的基本过程和过程中涉及的主要组件。

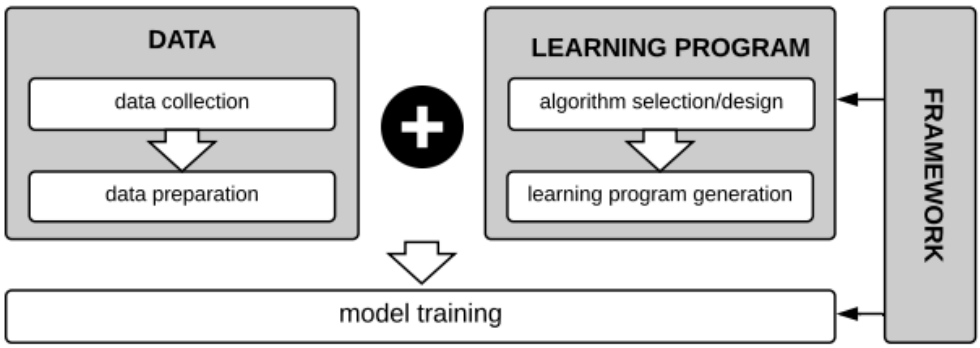

图 3: 构建 ML 模型的基本过程和过程中涉及的主要组件

因此，在进行 ML 测试时，开发人员可能需要尝试在每个组件中找到 bug，包括数据、学习程序和框架。其中，误差传播是一个比较严重的问题。与传统软件相比，ML 开发中的组件之间的联系更加紧密，这表明了测试每个 ML 组件的重要性。下面我们将介绍每个 ML 组件中的 bug 检测：

1. **Bug Detection in Data.** ML 模型 training 非常依赖于数据集。数据中的 bug 会影响生成的模型的质量，并且在一段时间内会被放大，产生更严重的问题。数据 Bug 的检测包括，对于训练和测试模型来说，数据是否充足，也就是数据是很完善；当前收集到的数据是否可以代表未来的数据，数据中是否包含噪声或者偏移较大的标签；是否在训练数据和测试数据之间有迁移；是否包含能够使模型性能变差的“有毒”的数据或对手信息。

2. **Bug Detection in Frameworks.** 机器学习需要大量的计算。ML 框架提供了编写学习程序的算法的帮助，以及帮助训练机器学习模型的平台，使开发人员更容易为复杂问题设计、训练和验证算法和模型建立解决方案。它们在 ML 开发中扮演着比传统软件开发更重要的角色。

3. **Bug Detection in Learning Program.** 学习程序可以分为两部分：由开发人员设计或从框架中选择的算法；以及开发人员为实现、部署或配置算法而编写的实际代码。学习程序中的错误可能出现，要么是因为算法的设计、选择或配置不当，要么是因为开发人员在实现设计的算法时犯了拼写错误或其他错误，使得程序没有实现想要设计的算法的功能。

4.4 ML testing Properties

测试属性是指在 ML 测试中要测试什么：对于一个训练好的模型，ML 测试需要保证什么条件。

本节列出了文献中考虑过的一些典型属性。我们将它们分为基本功能需求 (即正确性和模型相关性) 和非功能需求 (即效率、健壮性、公平性、可解释性)。当考虑到根本原因时，这些性质并不是严格地相互独立的，但它们是 ML 系统行为的不同外部表现，值得在 ML 测试中独立处理。

4.4.1 Correctness

正确性衡量的是被测的 ML 系统“得到正确结果”的概率。**(Definition 3) Correctness:** 令 $\mathcal{D}$ 为未来未知数据的分布。设 x 是属于 $\mathcal{D}$ 的数据项。令 h 为需要被测试的机器学习模型。$h(x)$ 是预测结果，$c(x)$ 为真实的 label。模型的 correctness $E(h)$ 为 $h(x)$ 和 $c(x)$ 一样的概率：

$$E(h) = \Pr_{x\sim\mathcal{D}}[h(x) = c(x)] \tag{1}$$

ML 系统的实际性能应该根据未来的数据进行评估。由于未来数据通常不可用，当前的最佳实践通常将数据分割为训练数据和测试数据 (或训练数据、验证数据和测试数据)，并使用测试数据来模拟未来数据。这种数据分割方法称为交叉验证。

4.4.2 Model Relevance

机器学习模型来自于机器学习算法和训练数据的结合。重要的是要确保所采用的机器学习算法不过于复杂。否则，模型对未来的数据可能没有很好的性能，或者有很大的不确定性。也就是通常所说的过拟合问题。

算法能力代表了机器学习模型可以选择的作为最优解的函数数量 (基于拥有的训练数据)。对于分类任务，通常用 vc-dimensional 或 Rademacher Complexity 来近似表示。vc 维是算法可以分开最大一组点的基数。Rademacher complexity 是最大的训练数据集的基数。

我们将机器学习算法能力与数据分布之间的相关性定义为模型相关性问题。

(Definition 5) Model Relevance: 令 $\mathcal{D}$ 为训练数据分布，$R(\mathcal{D},\mathcal{A})$ 是任何机器学习算法 $\mathcal{A}$ 对 $\mathcal{D}$ 最简单的要求的能力。$R'(\mathcal{D},\mathcal{A}')$ 是关于机器学习算法 $\mathcal{A}'$。Model Relevance 被定义为：

$$f = |R(\mathcal{D},\mathcal{A}) - R'(\mathcal{D},\mathcal{A}')| \tag{2}$$

模型相关性旨在衡量机器学习算法与数据的匹配程度。模型相关性低通常是由于过拟合造成的，其中模型对数据来说过于复杂，因此无法推广到未来的数据的应用上。当然，R 的计算没那么的简单，大多时候只能近似。

4.4.3 Robustness

因此，鲁棒性度量了 ML 系统在存在扰动时的，能正确性的执行任务的能力。
(Definition 6) Robustness: 令 $\mathcal{S}$ 为一个机器学习系统；$E(\mathcal{S})$ 为其正确性；假设 $\delta(S)$ 是对任何机器学习组件 (如数据、学习程序或框架) 都有扰动的机器学习系统。鲁棒性被定义为：

$$r = E(s) - E(\delta(S)) \tag{3}$$

鲁棒性研究的一个子类别称为对抗性鲁棒性。为了对抗的健壮性，扰动被设计成难以检测的形式。我们将对抗鲁棒性分为局部对抗鲁棒性和全局对抗鲁棒性。局部对抗鲁棒性定义如下。
(Definition 7) Local Adversarial Robustness: 令 x 为 ML 模型 h 的输入，设 x' 是对 x 进行对抗性扰动产生的另一个测试输入。模型 h 为 δ 局部鲁棒为，

$$\forall x' : \|x - x'\|_p \leq \delta \rightarrow h(x) = h(x') \tag{4}$$

局部对抗鲁棒性涉及一个特定测试输入的鲁棒性，而全局对抗鲁棒性度量针对所有输入的鲁棒性。我们定义全局对抗鲁棒性如下。**(Definition 8) Global Adversarial Robustness:**

$$\forall x, x' : \|x - x'\|_p \leq \delta \rightarrow h(x) - h(x') \leq \epsilon \tag{5}$$

有点类似连续和一直连续的概念。

4.4.4 Security

ML 系统的安全性是系统操作或非法访问 ML 组件所造成的潜在伤害、危险或损失。安全性和健壮性密切相关。鲁棒性较低的 ML 系统可能是不安全的：如果它在抵御预测数据中的扰动方面鲁棒性较差，系统可能更容易成为敌对攻击的受害者；例如，如果它在抵抗训练数据扰动方面不够稳健，它也可能容易受到数据污染 (即敌对地修改训练数据导致预测行为的变化)。

4.4.5 Data Privacy

机器学习中的隐私是机器学习系统保存隐私数据信息的能力。对于正式定义，使用目前最流行的差异隐私。
(Definition 9) $\epsilon-$Differential Privacy: 令 $\mathcal{A}$ 为随机算法。设 D_1 和 D_2 是在一个实例上不同的两个训练数据集。设 $\mathcal{S}$ 是 $\mathcal{A}$ 的输出集的子集。$\mathcal{A}$ 具有 $\epsilon-$Differential Privacy 的条件为：

$$\Pr[\mathcal{A}(D_1) \in S] \leq \exp(\epsilon) * \Pr[\mathcal{A}(D_2) \in S] \tag{6}$$

实际上，即为来知道任何一个人的数据是否对结果有重大影响。目前的研究主要集中在如何提出可以保护隐私的机器学习方法，而不是检测隐私侵犯。

4.4.6 Efficiency

机器学习系统的效率是指它的构建速度或预测速度。当系统在构建或预测阶段执行缓慢甚至无限时，就会出现效率问题。随着数据的指数级增长和系统的复杂性，效率是模型选择和框架选择要考虑的一个重要特征，有时甚至比准确性更重要。

例如，要将大型模型部署到移动设备上，可能需要执行优化、压缩和面向设备的定制，以便在合理的时间内使移动设备可执行，但可能会牺牲准确性。

4.4.7 Fairness

机器学习是一种统计方法，被广泛应用于决策，如收入预测、医疗预测等。机器学习倾向于学习人类教它的东西 (即以训练数据的形式)。然而，人类可能对认知有偏见，进一步影响收集或标记的数据和设计的算法，导致机器学习系统的偏见问题。

敏感且需要保护的特性被称为敏感属性。法律认可的敏感属性包括种族、肤色、性别、国籍、国籍、年龄、怀孕、家庭状况、残疾状况、退伍军人状况和基因信息。

公平性通常是特定领域考虑的，包括信贷、教育、就业、住房和公共住房等。制定公平是解决公平问题和建立公平机器学习模型的第一步。文献中提出了许多公平的定义，但目前尚未达成坚定的共识。

4.4.8 Interpretability（可解释性）

机器学习模型通常用于医疗、收入预测或个人信用评估方面的决策。对人类来说，理解最终决策背后的逻辑可能很重要，这样才可以建立对 ML 所做决策的信任。

可解释性的动机和定义各不相同，且仍有些不一致。然而，与公平性不同的是，对 ML 可解释性的数学定义仍然难以捉摸。参考前人的工作，文章将 ML 的可解释性描述为观察者能够理解 ML 系统做出的决策的原因的程度。

可解释性包括两个方面：透明度 (模型如何工作) 和事后解释 (可以从模型中派生的其他信息，来理解如何做出的决策)。GDPR 等法规也将可解释性视为一种必须的要求，用户有权要求解释和他们相关的算法做出的决策。

可能除了隐私性不太好理解之外，其他的还挺好理解的。

4.5 Software Testing vs. ML Testing

本节描述的是传统的软件测试与机器学习测试的差异。

1. **组件测试** (Bug 可能存在的地方)：传统的软件测试检测代码中的缺陷，而 ML 测试检测数据、学习程序和框架中的缺陷。

2. **被测行为**：传统软件通常是在需求固定的时候就固定的，而 ML 模型的行为可能会随着训练数据的更新而频繁变化。

3. **测试输入**：传统的软件工程在测试阶段通常是输入数据。这里需要区分一下测试输入和测试数据的概念，测试输入指的是指的是可以用来进行机器学习测试的任何形式的输入，而测试数据专门指用于验证 ML 模型行为的数据。测试学习程序时，测试用例可以是测试数据中的单个测试实例；当测试数据时，测试输入可以是一个学习程序。

4. **测试预期**：传统的软件测试，开发人员可以根据预期值验证输出，因此 oracle 通常是事先确定的。然而，机器学习被用来根据一组输入值生成预测值。生成的大量预测结果的正确性通常是手动确认的。即使对于一个具体的领域特定的问题，oracle 识别仍然是耗时和劳动密集型的，因为通常需要领域特定的知识。也就是打标签很困难。

5. **测试是否充足：**测试充分性标准用于评价测试是否完善。到目前为止，许多充分性标准被提出，例如，线路覆盖，分支覆盖，数据流覆盖。然而，由于机器学习软件的编程范式和逻辑表示格式与传统软件存在根本性的差异，需要新的测试充分性标准来适应机器学习软件的特点。

6. **检测 bug 中的假阳性**：由于难以获得可靠的 oracle, ML testing 往往在报告的 bug 中产生更多的假阳性。

7. **测试人员的角色**：ML 测试中的 bug 可能不仅存在于学习程序中，也可能存在于数据或算法中，因此数据科学家或算法设计者也需要扮演测试人员的角色。

5 ML Testing Workflow

有关于 ML testing workflow 的研究和图二中的流程相似。ML testing 包括 offline testing 和 online testing。

5.1 Test Input Generation

本小节主要研究如何生成测试输入。

5.1.1 特定领域测试输入生成

ML testing 的测试输入可以分成两种：对抗输入和自然输入。对抗性输入在原始输入的基础上进行扰动，它们可能不属于正常的数据分布 (例如，在实践中可能很少存在)，但可能暴露系统的鲁棒性或安全缺陷。相反，自然输入是那些属于实际应用场景的数据分布的输入。在这里，我们将介绍通过特定领域的测试输入，来生成自然输入的相关工作。

DeepXplore (SOSP 2017) 提出了一种白盒差分 testing 技术来生成 deep learning system 的测试输入。受传统软件测试中测试覆盖率的启发，作者提出了神经元覆盖率来驱动测试生成。我们希望测试输入具有较高的神经元覆盖率。此外，输入需要暴露不同 DNN 模型之间的差异，并尽可能接近真实数据。联合优化算法迭代地使用梯度搜索来找到一个满足所有这些目标的输入。

为了让自动驾驶系统创建有用和有效的数据，DeepTest(ICSE 2018) 对 9 种不同的真实图像变换进行了贪婪搜索: 改变亮度、改变对比度、平移、缩放、水平剪切、旋转、模糊、雾效果和雨效果。OpenCV 提供了三种类型的图像转换样式：线性、仿射和卷积。DeepTest 的评估使用的是 Udacity 自动驾驶汽车挑战数据集。它在 cnn 和 rnn 上检测到超过 1000 种错误行为，且假阳性率很低。

GAN 是一种生成模型，并且 GAN 曾经成功的应用在图片迁移领域。Zhang (ASE 2018) 将 GAN 应用于各种天气条件下基于驾驶场景的测试生成。他们从 Udacity Challenge 数据集和 YouTube 视频 (下雪或下雨的场景) 中取样图像，并将它们输入 UNIT 框架 (一种图片迁移的网络) 进行训练。训练后的模型将整个 Udacity 图像作为输入，并将迁移后的图像作为生成的测试样例。

为了测试基于音频的深度学习系统，Du 等人设计了一套针对音频输入的迁移，考虑了背景噪声和音量变化。他们首先从 RNN 中提取概率转移模型。在此基础上，定义了有状态测试准则，并用于指导状态机器学习系统的测试生成。

Ding 等 (SCC 2019) 在对生物细胞图像进行分类时，为了测试图像分类平台，构建了一个生物细胞分类器测试框架。该框架迭代生成新的图像，并使用变性关系进行测试。例如，他们通过在生物细胞图像中增加人工线粒体的数量/形状来生成新的图像，这可以产生容易识别的变化的分类结果。

5.1.2 模糊和基于搜索的测试输入生成

模糊测试是一种传统的自动测试技术，它生成随机数据作为程序输入，以检测崩溃、内存泄漏、失败 (内置) 断言等。作为另一种广泛应用的测试生成技术，基于搜索的测试生成通常使用元启发式搜索技术来指导模糊过程，从而使测试生成更加高效另一种广泛应用的测试生成技术，基于搜索的测试生成，通常使用元启发式搜索技术来指导模糊过程，从而使测试生成更加高效。这两种技术在探索 ML testing 输入空间方面也被证明是有效的:

Odena 等人提出了 TensorFuzz，使用了一种简单的最近邻爬山方法来探索 Tensorflow 图在有效输入空间上实现覆盖，并发现数值错误、神经网络及其版本之间的不一致，以及在 rnn 中出现不良行为。

DLFuzz 由 Guo 等人提出 (FSE 2018)，是另一种基于以 neuron coverage 为基础思想，实现 DeepXplore 的模糊测试生成方法。DLFuzz 的目的是生成对抗样本。因此，生成过程不需要类似的功能性深度学习系统来进行交叉引用检查，如 DeepXplore 和 TensorFuzz。相反，它只需要对原始输入进行最小的改变，就可以找到那些提高神经元覆盖但与原始输入具有不同预测结果的新输入。对 MNIST 和 ImageNet 的初步评估表明，与 DeepXplore 相比，DLFuzz 能够多产生 135% - 584.62% 的输入，减少 20.11% 的时间消耗。

Wicker 等人 (ICTACAS) 提出了特征引导的测试生成。他们采用尺度不变特征变换 (SIFT) 来识别代表高斯混合模型图像的特征，然后将寻找对抗样本的问题转化为基于回合制的两方随机博弈。并使用蒙特卡罗树搜索来识别图像中最脆弱的那些元素，作为生成对抗样本的方法。实验表明，他们的黑盒方法可以与一些先进的白盒方法相媲美。

Uesato 等人 (ICLR 2019) 提出通过生成对抗样本来评估强化学习。灾难性故障的检测成本非常的昂贵的，因为故障很少发生。为了减少发现此类故障的后续成本，作者提出使用故障概率预测器来估计 agent 发生故障的概率。

除了图像分类之外，还有用于特定应用场景的模糊器。Zhou 等将模糊和变形测试相结合，测试了自动驾驶汽车的 LiDAR 障碍感知模块，并报告了之前未知的软件故障。

研究了如何通过将故障建模为贝叶斯网络来生成最有效的测试用例 (最可能导致违反安全条件的故障)。该评估基于英伟达和百度的两款生产级 AV 系统，揭示了许多因故障导致违反安全的情况。

Udeshi 和 Chattopadhyay 为文本分类任务生成输入，并提出一种考虑被测语法和输入之间距离的模糊方法。Nie 等人和 Wang 等人修改了 NLI(自然语言推理) 任务中的句子，以生成用于鲁棒性测试的测试输入。Chan 等人生成了 DNC 的对抗例子，以揭示其鲁棒性问题。Udeshi 等人 (ICASE) 非常关注个体的公平性，并生成了强调被测模型是否存在歧视性的测试输入。

Tuncali 等人 (IV 2018) 提出了一个测试自动驾驶系统的框架。在他们的工作中，他们比较了三种测试生成策略：随机模糊测试生成，覆盖数组 + 模糊测试生成，以及覆盖数组 + 基于搜索的测试生成 (使用模拟退火算法)。结果表明，采用基于搜索技术的测试生成策略效果最好。

5.1.3 Symbolic Execution Based Test Input Generation

符号执行是一种程序分析技术，用于测试被测软件是否违反某些确定属性。动态符号执行 (DSE，也称为 concolic 测试) 是一种用于自动生成测试输入以实现高代码覆盖率的技术。DSE 使用随机测试输入测试程序，并并行执行 Symbolic Execution，以收集从分支语句中的谓词获得的符号约束。沿着一条路径的所有符号约束的结合称为路径条件。在生成测试时，DSE 从输入域中随机选择一个测试输入，然后使用约束求解在路径中达到一个目标分支条件。在 ML 测试中，模型的性能不仅由代码决定，还由数据决定，因此 Symbolic Execution 有两种应用场景：数据和代码。

然而对机器学习代码使用 Symbolic Execution 有很多的挑战。Gopinath 在他们的论文中列出了神经网络面临的三个的挑战，这些挑战也适用于其他 ML 算法: (1) 网络没有明确的分支;(2) 网络可能是高度非线性的，没有完善的求解约束;(3) 由于 ML 模型的结构通常非常复杂，超出了当前符号推理工具的能力，因此存在可扩展性问题。

也有很多解决这些挑战的方法，Gopinath 引入了 DeepCheck。DNN 转换为一个程序，使 Symbolic Execution 找到与原始图像具有相同的激活模式的像素攻击方法。特别是 DNN 中的激活功能遵循 IF-Else 分支结构，这可以看作是通过翻译程序的路径。DeepCheck 可以通过识别大部分神经网络无法正确识别的图像的像素或像素对来创建 1 像素和 2 像素攻击。

类似地，Agarwal 等人应用 LIME(SIGKDD 2016)，这是一种用线性模型、决策树来近似模型的局部解释工具，以帮助获得 Symbolic Execution 中使用的路径。他们基于 8 个 baselines 的评估表明，该算法生成的成功测试用例是 THEMIS 随机测试生成方法的 3.72 倍。

un 等人 (ASE 2018) 提出了一种动态的 DNN Symbolic Execution 测试方法 DeepConcolic。此方法对 ML 模型的给定属性进行具体的评价，从而对特定的 MC/DC 准则条件进行符号分析。DeepConcolic 明确地将覆盖率要求作为输入。作者报告说，与 DeepXplore 评估的模型相比，它能产生超过 10% 的神经元覆盖率。

5.2 Test Oracle

Test oracle 是 ML 测试中的关键问题之一，其为了判断一个 bug 是否存在。这就是所谓的 “Oracle Problem”。在 ML testing 中，oracle 问题很难解决，因为许多机器学习算法是概率程序。在本节中，我们列出了几种流行的测试 oracle 类型，例如，metamorphic 关系、交叉引用和模型评估度量。

5.2.1 Metamorphic Relations as Test Oracles

变质关系 (Metamorphic Relations) 的提出数为了改善传统软件 testing 中的 test oracle 问题。变质关系是指在多个程序执行期间，软件输入变化和输出变化之间的关系。比如，为了测试函数 $\sin(x)$，可以检查当输入从 x 改为 $\pi - x$ 时函数的输出是如何变化的。如果 $\sin(x)$ 和 $\sin(\pi - x)$ 不同，表示存在错误。$\sin(x) = \sin(\pi - x)$ 之间是 Metamorphic Relations 可以帮助检测 Bug。

在机器学习测试中，Metamorphic Relations 被广泛研究以解决 oracle 问题。许多 Metamorphic Relations 是基于训练或测试数据的转换，这些转换预期会在预测输出中产生不变的或特定的预期变化。在研究相应的 Metamorphic Relations 时，数据转换具有不同的粒度。有些转换执行粗粒度的更改，例如扩大数据集或更改数据顺序，而不更改每个数据实例。我们将这些转换称为粗粒度数据转换。有些转换通过对每个数据实例进行较小的更改来进行数据转换，例如改变图像的属性、标签或像素，在本文中将这些转换称为细粒度数据转换。下面介绍每种类型转换的相关工作。

Coarse-grained Data Transformation:

最常用的六种粗粒度的数据转换方法：介绍了输入数据的六种转换方法：加法、乘法、置换式、反式、包容式和互斥式。这些变化包括在数值中添加一个常数；将数值与常数相乘；对输入数据的顺序进行排列；颠倒输入数据的顺序；删除输入数据的一部分；添加额外的数据。

Ding 等人提出了 11 种 Metamorphic Relations 来测试深度学习系统。在数据层面，Metamorphic Relations 也基于训练数据和测试数据转换，不应该影响分类精度，如添加 10% 训练图像的每个类别训练数据集或删除数据集的某个数据类别。研究的是基于生物细胞图像的分类。

Fine-grained Data Transformation:

2009 年，Xie 等人提出使用特定于某一模型的 Metamorphic Relations 来测试监督分类器。本文介绍了五种类型的 Metamorphic Relations，它们能够根据输入的特定变化预测输出的预期变化 (如类、标签、属性的变化)。对 KNN 和朴素贝叶斯实现的分析表明，并非所有的 Metamorphic Relations 都是必要的。支持向量机和神经网络在 Metamorphic Relations 上的差异也有文章进行了讨论。Dwarakanath 等人 (SIGSOFT 2018) 利用支持向量机和深度学习系统将 Metamorphic Relations 应用于图像分类。数据上的变化包括改变特征或实例顺序，测试特征的线性缩放，测试数据的归一化或缩放，或改变数据的卷积运算顺序。所提出的 Metamorphic Relations 能够找到 71% 的 Bug。Sharma 和 Wehrheim 考虑了细粒度数据转换，例如更改特性名称、重命名特性值以测试公平性。他们研究了 14 种分类器，没有一种对特征名称变换敏感。

Zhang 等人提出了扰动模型验证 (perturturmodel Validation, PMV)，该方法结合了 Metamorphic Relations 和数据突变来检测过拟合。PMV 通过在训练数据中注入噪声来对训练数据进行变异，建立扰动训练数据集，然后检查噪声程度增加时训练准确率下降率。训练精度下降得越快，机器学习模型过拟合越少。

Tian 等人 (ICSE 2018) 和 Zhang 等人 (ASE 2018) 指出，在不同天气条件下，自动驾驶汽车对同一个场景不应出现明显的转向角变化或保持不变。Ramanagopal 等利用相似图像的分类一致性作为测试自动驾驶汽车的 test oracles。评估表明，在检测未标记数据中的错误时，精度为 0.94。

此外，Xie 等人提出了 METTLE，一种用于无监督学习验证的 Metamorphic testing 方法。METTLE 有六种不同粒度的 Metamorphic Relations，它们是专门为无监督学习设计的。这些 Metamorphic Relations 操纵实例顺序、特殊性、密度、属性，或注入数据的异常值。该评价基于 Scikit-learn 生成的综合数据。Nakajima 等人 (APSEC 2018) 讨论了使用不同粒度的 Metamorphic Relations 在 SVM 和神经网络中寻找问题的可能性，例如操纵实例顺序或属性顺序，反向标签和改变属性值，或操纵图像中的像素。

Metamorphic Relations between Different Datasets:

不同数据集之间的一致性关系也可以看作是 Metamorphic Relations，用来检测数据 bug。Kim 等和 Breck 等 (SysML) 研究了训练数据和新数据之间的 Metamorphic Relation。如果训练数据和新数据有不同的分布，训练数据可能不够充分。Breck 等人也研究了时间相近的不同数据集之间的 Metamorphic Relations：实际预计这些数据集具有一些共同的特征，因为数据生成代码很少频繁发生大的改变。

Frameworks to Apply Metamorphic Relations:

Murphy et al(ISSTA 2019) 提出了 Amsterdam 框架其中使用 Metamorphic Relations 来自动检测 ML 中的 bug。他们还开发了 Corduroy，它扩展了 Java 建模语言，让开发人员指定 metamorphic 属性并为 ML 测试生成测试用例。

5.2.2 Cross-Referencing as Test Oracles

交叉引用是另一种用于 ML 测试的测试 oracle，包括差异测试和 n 版本编程。差异测试是一种传统的软件测试技术，它通过观察相似的应用程序对于相同的输入是否产生不同的输出来检测 bug，是检测编译器 bug 的测试 oracle(SIGPLAN 2014)。根据 Nejadgholi 和 Yang(ASE 2019) 的研究，深度学习库中 5% 到 27% 的测试 oracle 使用了差分测试。

差异测试和 n 版本编程之间联系紧密。n 版本编程是用多种方法来实现同一个功能，从而使不同版本的组合具有更强的容错性和健壮性。

Davis 和 Weyuker 讨论了对不可测试程序进行差异测试的可能性。其思想是，如果一个算法的多个实现对一个相同的输入产生不同的输出，那么至少有一个实现包含缺陷。Alebiosu 等人 (AAAI, EDSMLS) 在机器学习中评估了这种想法，并成功地从 7 种朴素贝叶斯实现中发现了 16 个错误，从 19 个 k-最近邻实现中发现了 13 个错误。

Pham 等人 (ICSE 2019) 也采用了交叉引用来测试 ML 实现，但侧重于深度学习库的实现。他们提出了 CRADLE，这是第一个专注于在深度学习软件库中寻找和定位 bug 的方法。评估使用了 3 个库 (TensorFlow、CNTK 和 Theano)、11 个数据集 (包括 ImageNet、MNIST 和 KGS Go game) 和 30 个预训练模型。结果表明，CRADLE 能够检测到 104 个不一致的地方和 12 个 bug。

DeepXplore(SOSP 2017) 和 DLFuzz(FSE 2018) 使用差分测试作为测试 oracle 来寻找有效的测试输入。在测试生成过程中，那些导致不同算法或模型之间的不同行为的测试输入是首选的。

Sun 等人 (ICSE 2020) 将交叉引用应用于机器翻译系统的修复。他们的方法，“TransRepair”，比较不同突变输入的输出，并选择与其他输入最相似的输出作为最好的翻译结果的候选。

5.2.3 Measurement Metrics for Designing Test Oracles

一些研究提出了 ML 系统非功能特征的定义或统计度量，包括鲁棒性、公平性和可解释性。这些度量并不是可以判断一个系统是不是鲁棒，公平或者可解释的，但是对于测试人员理解和评估测试的特性非常重要，并提供一些可以与预期相比较的实际统计数据。例如，不同的对公平性的定义定义了 ML 系统必须满足的条件，如果没有这些条件，系统就不公平。这些定义可以直接用于检测违反公平的行为。

5.3 Test Adequacy

测试充分性评价旨在发现现有测试是否具有良好的缺陷揭示能力。它提供了测试活动的客观置信度度量。充分性标准也可以用来指导测试生成。传统软件测试中常用的测试充分性评估技术包括代码覆盖测试和变异测试，这些技术在 ML 测试中也被采用。

5.3.1 Test Coverage

在传统的软件测试中，代码覆盖率表示程序源代码被测试方法执行的程度 (ISSTA 2016)。测试方法的覆盖率越高，隐藏的 bug 就越有可能被发现。换句话说，覆盖代码片段是检测隐藏在代码中的缺陷的必要条件。通常需要创建测试方法实现更高的覆盖率。与传统软件不同，代码覆盖率很少是 ML 测试的要求标准，因为 ML 模型的决策逻辑不是手动编写的，而是从训练数据中学习的。例如，在 Pei 等人 (ICSE 2017) 的研究中，100% 的传统代码覆盖率很容易通过单个随机选择的测试输入实现。**所以，研究人员为 ML 模型提出了代码覆盖之外的各种类型的覆盖**。

Neuron coverage.

Pei 等人 (ICSE 2017) 提出了第一个覆盖标准，神经元覆盖，用于深度学习测试。神经元覆盖率计算方法为，测试激活的神经元数量与 DNN 中神经元总数的比值。如果一个神经元的输出值大于指定的阈值，则表明被激活。

Ma 等人 (ASE 2018) 扩展了神经元覆盖的概念。他们首先根据训练数据构建 DNN，以便根据训练数据获得每个神经元的激活行为。基于此，他们提出了更细粒度的标准，k-multisection 神经元覆盖、神经元边界覆盖和强神经元激活覆盖，来代表 DNN 的主要功能行为和边缘行为。

MC/DC coverage variants.

Sun 等人受 MC/DC 覆盖标准的启发，提出了四种测试覆盖标准，适用于 DNN 的不同特征。MC/DC 观察一个布尔变量的变化，比如察一个神经元的符号、值或距离的变化，以捕获测试输入中的因果变化。该方法假定 DNN 是一个完全连接的网络，不考虑其自身层内神经元的环境以及同一层内不同神经元的组合。

Layer-level coverage.

Ma 等人 (ASE 2018) 也提出了层级覆盖标准，该标准考虑最活跃的神经元及其组合来表示 DNN 的行为。基于数据集 MNIST 和 ImageNet 对神经元覆盖进行了评价，认为该覆盖具有更好的性能。在随后的工作中，他们进一步提出了组合测试覆盖，通过检查一层神经元激活交互作用的比例来检查每层神经元的组合激活状态。Sekhon 和 Fleming 定义了一个覆盖标准，即寻找 1) 同一层中的所有神经元对，具有所有可能出现的值组合，2) 连续层中的所有神经元对，所有可能的值组合。

State-level coverage.

虽然上述准则在一定程度上捕捉了前馈神经网络的行为，但它们并没有像递归神经网络 (RNN) 那样明确地刻画有状态机器学习系统的特征。基于 rnn 的 ML 方法在处理连续输入 (如语音、自然语言、网络物理控制信号) 方面取得了显著的成功。为了分析这种有状态的 ML 系统，Du 等人提出了第一套专门用于基于 rnn 的有状态深度学习系统的测试标准。首先将一个有状态深度学习系统抽象为一个概率转移系统。他们提出了基于状态和转移模型的轨迹跟踪的标准，以捕获动态状态转换行为。

Limitations of Coverage Criteria.

本小节描述的是覆盖性测试标准的局限性。目前大多数覆盖标准都是基于 DNN 的结构。Li(ICSE NIER track) 中提出，对深度网络结构性覆盖标准的主要局限是神经网络和人写程序之间的根本区别。他们实验发现，测试集中错误分类输入的数量，与 DNN 结构性覆盖的结论之间没有很强的相关性。这是由于机器学习系统的黑盒特性，目前尚不清楚这些标准与系统的决策逻辑之间的关系。

5.3.2 Mutation Testing

在传统的软件测试中，突变测试通过加入故障来评估测试方法的故障显示能力 (ISSTA 2016, TSE 2011)。检测到的故障与所有注入的故障的比例称为突变得分。在机器学习测试中，机器学习系统的行为不仅依赖于代码，还依赖于数据和模型结构。Ma 等人 (ISSRE 2018) 提出了 DeepMutation 算法，该算法在源码或模型上突变 DNN，对 DNN 的决策边界进行微小扰动。在此基础上，变异分数被定义为，输出结果与更改之前不同的测试实例，与实例总数的比例。

与结构覆盖标准相比，基于突变检测的标准与 DNN 的决策边界更直接相关。例如，接近 DNN 的决策边界的输入数据可以更容易地检测出 DNN 与其突变行为之间的不一致性。

5.3.3 Surprise Adequacy

Kim 等人 (ICSE 2019) 引入了 Surprise Adequacy 来衡量深度学习系统中离散输入意外范围的覆盖率。他们认为，测试多样性在与训练数据相比较时更有意义。一个“好的”测试输入应该与训练数据相比“充分但不过分 surprising”。引入了两种 Surprise 测量方法：一种是基于核密度估计 (KDE, Kernel Density Estimation 感觉作者这里把 Kernel 拼错了) 来近似系统在训练过程中看到类似输入的可能性；另一种是基于表示给定输入的神经元激活痕迹的向量和训练数据之间的距离 (例如，欧氏距离)。这些标准可用于检测对抗性的输入。需要进一步的研究来确定这些标准是否能使 ML 模型的行为边界近似于惊讶程度。对于未来的工作来说，研究对抗样本、自然误差样本和基于意外的标准之间的关系也是很重要的。**我理解的这里的意外充足性指的是有没有尽量的考虑到程序遇到的特殊数据**。

5.3.4 Rule-based Checking of Test Adequacy

Breck 等人提出了 28 个测试方面和谷歌使用的评分系统。他们的重点是衡量一个给定的机器学习系统的测试情况。28 个测试指标分为四种类型:1) 对 ML 模型本身的测试，2) 对用于构建模型的 ML 基础设施的测试，3) 对用于构建模型的 ML 数据的测试，4) 检查 ML 系统是否随时间正确工作的测试。它们中的大多数是一些必须检查的规则，可以应用于指导测试生成。例如，训练过程应是可重复的；所有功能都应该是有益的；没有其他模型比当前的模型更简单，但性能更好。研究表明，尽管 ML 测试是复杂的，但设计一些基本的测试用例来测试 ML 系统的基本功能是有共性的。

5.4 Test Prioritisation and Reduction

ML 中的测试输入生成需要非常大的输入空间。并且，需要标记每个测试实例，以判断预测的准确性。这两个方面导致测试产生成本较高。生成测试输入在计算上也是昂贵的。Zhang 等人提出通过识别那些表示更有效的对抗性样本的测试实例来降低成本。该方法是一种测试优先级技术，它根据测试实例对噪声的敏感性对它们进行排序，因为对噪声更敏感的实例更可能产生对抗性样本。

Li 等人 (FSE 2019) 专注于在 DNN 测试中减少测试数据。他们提出了一种基于交叉熵最小化分布近似的采样技术，该方法由 DNN 最后一隐藏层的神经元引导。评估三个图像数据集:MNIST、Udacity challenge 和 ImageNet 对预训练模型进行的。结果表明，与随机抽样相比，他们的方法样本仅为测试输入的一半，但达到了类似的精度水平。

Ma 等人 (TOSEM 2021) 提出了一套基于模型置信度概念的测试选择度量方法。对模型来说不确定的测试输入是更重要的，因为它们信息量更大，应该重点考虑用于改进模型。评价结果表明，他们的测试选择方法比随机选择方法有 80% 以上的增益。

5.5 Bug Report Analysis

Thung 等人通过分析机器学习系统的 bug 报告来研究机器学习 bug。结果表明，22.6% 的 bug 是由于所定义的算法的错误实现造成的。此外，15.6% 的 bug 是非功能性 bug。5.6% 的 bug 是数据 bug。

Zhang 等人 (SIGSOFT 2018) 基于 Github 或 StackOverflow 的 bug 报告，研究了 175 个 TensorFlow 的 bug。他们研究了 Bug 的症状和根本原因，现有的 Bug 检测挑战以及如何处理这些 Bug。他们将 TensorFlow bug 分为异常或崩溃、低正确性、低效率和未知。造成 Bug 的主要原因是算法设计和实现，如 TensorFlow API 误用 (18.9%)，张量未对齐 (13.7%)，模型参数或结构错误 (21.7%)。

Banerjee 等人 (DSN 2018) 分析了 12 家自动驾驶汽车制造商的自动驾驶系统 bug 报告，在加州累计行驶了 1,116,605 英里。他们使用 NLP 技术将 disengagements 的原因分为 10 种类型 (disengagements 是指自动驾驶控制的行为和人类驾驶员的不同的地方)。根据他们的报告分析，机器学习系统和决策控制方面的问题是导致 disengagements 的主要原因。

5.6 Debug and Repair

Data Resampling.

前面部分介绍了生成的测试输入只暴露了 ML 错误，但也可以作为训练数据的一部分进行了研究，可以通过再训练提高模型的正确性。例如，DeepXplore 通过用生成的输入重新训练深度学习模型，实现了高达 3% 的分类精度提高。DeepTest(ICSE 2018) 将模型的准确率提高了 46%。Ma 等人 (TOSEM 2021) 找到导致错误分类的神经元，称其为错误神经元。他们对影响这些错误神经元的训练数据进行了重新采样，以帮助改善模型的表现。

Debugging Framework Development.

Dutta 等人提出了 Storm，这是一个程序转换框架，用于生成支持机器学习测试调试的小程序。为了修复 bug，开发人员通常需要缩小测试中的程序，以编写更好的 bug 报告，并方便调试和回归测试。Storm 运用程序分析和概率推理来简化概率程序，有助于更容易地找出问题所在。Vartak 等人 (ICDM 2018) 提出了 MISTIQUE 系统来捕获、存储和查询模型中间体，以帮助调试。

5.7 General Testing Framework and Tools

还有一些工作专注于提供测试工具或框架，帮助开发人员实现测试活动。有一个测试框架用于生成和验证安全测试的测试输入。Dreossi 等人提出了一种由三个主要模块组成的 CNN 测试框架：一个图像生成器、一组采样方法和一套可视化工具。Tramer 等人提出了一种全面的测试工具，以帮助开发人员通过易于解释的错误报告来测试和调试错误。Nishi 等人提出了一个包括许用性、可实现性、稳健性、可避免性和可改进性等不同评价方面的测试框架。还讨论了不同级别的 ML 测试，如系统、软件、组件和数据测试。

Thomas 等人 (Science 2019) 最近提出了一个设计机器学习算法的框架，它简化了不希望的行为的规则。该框架被证明适用于调节回归，分类和强化算法。它允许人们从 (可能有偏见的) 数据中学习，同时保证模型在应用于未见过的数据时，极有可能不会显示出偏见。用户可以对偏见进行各种各样的定义。对于学习算法和训练数据，框架要么返回一个保证没有偏见的模型，要么返回一个警告，说明它没有找到保证没有偏见的模型。

6 ML PROPERTIES TO BE TESTED

6.1 Correctness

正确性的概念有多重要，大家都知道。经典的机器学习验证方法是交叉验证和 bootstrap。其原理是通过数据采样分离测试数据，以检验训练的模型是否适合新的案例。有几种方法可以执行交叉验证。在 hold out 交叉验证中，数据被分成两部分：一部分成为训练数据，另一部分成为测试数据。在 k 倍交叉验证中，数据被分成 k 个相同大小的子集：一个子集作为测试数据，其余的 $k-1$ 个子集作为训

练数据。然后这个过程重复 k 次，每个子集都作为测试数据。在 Bootstrapping 中，数据采用替换抽样，因此测试数据可能包含重复的实例。这都是非常常见的采样方法。

目前有几个被广泛使用的正确性指标，如精确度、精度、查全率和曲线下面积 (AUC)。已有研究分析了每种测量标准的缺点。例如，准确性并不区分它所犯错误的类型 (假阳性和假阴性)。所以，我们应该仔细地选择性能指标。Chen 等人在评估 ML 分类器的正确性时研究了训练数据和测试数据的可变性 (PR)，他们推导了估计性能方差的解析表达式。他们还研究了不同统计方法在比较 AUC 时的表现，发现 f 检验的表现最好。

Qin 等人 (QRS) 提出从训练数据中生成一个镜像程序，然后利用这个镜像程序的行为作为正确性 oracle。镜像程序期望具有与测试数据类似的行为。

6.2 Model Relevance

模型相关性评估检测模型的复杂度和数据量之间是否匹配。模型相关性差通常与过拟合或欠拟合有关。交叉验证传统上被认为是一种检测过拟合的有效方法。然而，它并不总是清楚多少过拟合是可接受的，交叉验证不太可能检测过拟合，如果测试数据不代表潜在的 unseen 的数据。

Zhang 引入了扰动模型验证 (Perturbed Model Validation, PMV) 来帮助模型选择。PMV 在训练数据中注入噪声，根据扰动后的数据重新训练模型，然后利用训练精度下降率检测过拟合/欠拟合。我们的直觉是，过度拟合的模型倾向于在训练样本中拟合噪声，而欠拟合的模型无论是否注入噪声都将具有较低的训练精度。因此，对于扰动数据，过拟合和欠拟合对噪声的敏感性较低，且相对于噪声程度的精度变换率较小。实验结果表明，与 10 倍交叉验证相比，PMV 具有更好的性能，为检测过拟合和欠拟合提供了更容易识别的信号。

ML 系统通常在部署完成后收集新的数据，并将其添加到训练数据中，以提高准确性。但是，不能保证测试数据代表未来收集到的数据。Werpachowski 等人 (ICLR 2019) 提出了一种通过从测试数据中生成对抗样本的过拟合检测方法。如果对抗性样本的重加权误差估计与原始测试集的误差估计有足够的不同，则检测出过拟合。

Kirk 提到过，我们可以使用训练时间来衡量 ML 模型的复杂性；最好选择正确率相同但训练时间相对较小的算法。

Ma 等人 (FSE 2018) 试图通过对训练数据重新采样来缓解过拟合问题。根据使用三组图像分类数据集进行的评估，他们的方法被发现平均将测试精度从 75% 提高到 93%。

6.3 Robustness and Security

6.3.1 Robustness Measurement Criteria

与正确性或过拟合不同，鲁棒性是机器学习系统的非功能性特征。测量鲁棒性的一种方法是在存在噪声的情况下检查系统的正确性 (ICLR 2018)；一个稳健的系统应该在有噪声的情况下保持性能。

Moosavi-Dezfooli 等人 (CVPR 2016) 提出了 DeepFool，量化其鲁棒性。Bastani 等人 (NIPS 2016) 提出了三个度量鲁棒性的指标: 1) 点态鲁棒性，即输入变化最小的分类器不具有鲁棒性；2) 对抗频率，表示改变一个输入改变分类器结果的频率；3) 对抗严重度，表示输入与其最近的对抗样本之间的距离。

Tjeng 等人 (ICLR 2019) 提出使用测试输入与其最近的对抗样本之间的距离来衡量鲁棒性。Ruan 等 (IJCAI 19) 基于测试数据提供了全局鲁棒性下界和上界来量化鲁棒性。

最近，Mangal 等人提出了概率鲁棒性的定义。他们的工作使用抽象解释来近似神经网络的行为，并计算出输入区域的过度近似，在那里网络可能表现出非鲁棒行为。Banerjee 等人探索了使用贝叶斯深度学习来建模深度神经网络内部错误的传播，从而在没有进行大量的错误插入的实验的情况下，用数学模型模拟神经网络对硬件错误的敏感性。

6.3.2 Perturbation Targeting Test Data

关于对抗样本的生成，本文并不打算完全涵盖它。相反，我们关注的是在传统软件测试和机器学习的交汇处，那些有希望成为未来研究成果的领域。Carlini 和 Wagner(S & P) 提出了使用距离度量来量化相似性的对抗样本生成方法。

对抗输入生成已被广泛用于测试自动驾驶系统的鲁棒性。Papernot 等设计了一个库来标准化对抗性实例构建的实现。他们指出，标准化对抗性样本生成是很重要的，因为没有“对抗样本构建的标准化”而构建的基准是无法相互比较的：比如一个好的结果是因为较好的鲁棒性，还是由对抗样本构造过程的差异引起的，说也说不清。

6.3.3 Perturbation Targeting the Whole System

针对整个系统的扰动，甚至是硬件层次上的扰动一般不是我们搞 AI 的研究的内容，这里不做过多的解释了。

6.4 Efficiency

Zhang 等对 Tensorflow bugreated artifact (from StackOverflow QA page and Github) 的实证研究发现，175 个 ML bug 中有 9 个 (5.1%) 属于效率问题。原因可能是效率问题很少发生，也可能是这些问题难以检测。Kirk 指出，在训练模型时，可以利用不同的机器学习算法的效率来比较它们的复杂度。Spieker 和 Gotlieb 研究了三种训练数据约简方法，其目标是在模型训练过程中，在原始训练数据集中找到具有相似特征的更小子集，从而提高建模速度，提高机器学习测试的速度。

6.5 Fairness

公平性是一个相对较新出现的非功能性特征。根据 Barocas 和 Selbst 的研究，造成不公平的主要原因有以下五种。

1. **倾斜样本：**一旦出现一些初始偏差，这种偏差可能随着时间的推移而恶化。
2. **受污染的样本：**由于人类的偏见，数据标签也会带有偏见。
3. **有限的特征：**特征可能信息较少或收集不可靠，在构建特征和标签之间的联系时误导模型。
4. **样本量的差异：**如果少数群体和多数群体的数据高度不平衡，ML 模型可能会使少数群体的数据较差。
5. **代理：**一些特征是敏感属性的代理 (例如，一个人居住的邻域)，即使排除敏感属性，也可能导致 ML 模型的偏差。

公平研究的重点是衡量、发现、理解和处理观察到的不同群体或个体在表现上的差异。这种差异与公平漏洞有关，它可能会冒犯甚至伤害用户，导致程序员和企业不信任、收入损失，甚至违反法律。

6.5.1 Fairness Definitions and Measurement Metrics

为了帮助量化 ML 公平性的形式化，我们使用 X 表示一组个体，Y 表示针对 X 中的每个个体进行决策时的真实标签集。设 A 为敏感属性集合，Z 为其余属性

1. Fairness Through Unawareness. Fairness Through Unawareness (FTU) 是指只要在决策过程中没有显式使用需要被保护的属性，算法就是公平的。这是一种相对低成本的定义和确保公平的方式。然而，有时 X 中的非敏感属性可能包含与敏感属性相关的信息，从而导致分歧。排除敏感属性也可能影响模型的准确性，导致预测结果的有效性降低。

2. 如果基于敏感属性选择的群体具有相同的决策结果概率，则被测模型具有群体公平性。群体公平有几种类型。

   (a) Demographic Parity: 统计结构均衡是一种流行的群体公平测量方法。它要求决策应该独立于受保护的属性。设 G_1 和 G_2 是属于 X 的两个 Group，其划分标准是敏感属性 $a \in A$。模型 h 满足 Demographic Parity 的标准是，$P\{h(x_i) = 1 \mid x_i \in G_1\} = P\{h(x_j) = 1 \mid x_j \in G_2\}$。

   (b) Equalised Odds: 这是另一种衡量公平性的指标，

   $$P\{h(x_i) = 1 \mid x_i \in G_1, Y = y_i\} = P\{h(x_j) = 1 \mid x_j \in G_2, Y = y_i\}.$$

3. Counter-factual Fairness. 当被保护属性转换为反事实值时，模型的输出保持不变，并且根据假定的因果模型对其他变量进行修正，即满足反事实公平。

4. Individual Fairness. Dwork 等人提出了一种使用任务特定相似性度量来描述应该被视为相似的个体对。Dwork 等人认为，具有个体公平性的 h 模型应该在相似的个体之间给出相似的预测结果，

   $$P\{h(x_i) \mid x_i \in X\} = P\{h(x_j) = y_i \mid x_j \in X\} \text{ iff } d(x_i, x_j) < \epsilon.$$

5. Analysis and Comparison of Fairness Metrics. 尽管现有的公平定义很多，但每一种都有其优缺点。哪种公平是最合适的仍然存在争议。因此，有一些工作调查和分析现有的公平指标，并比较他们的表现基于实验结果，如下所述。Gajane 和 Pechenizkiy 调查了在文献中公平是如何定义和形式化的。Corbett-Davies 和 Goel 研究了三种类型的公平性定义。他们举例指出每种类型在统计学上的局限性。

   Verma 和 Rubin 解释并说明了基于公共统一数据集的现有最突出的公平性定义。Saxena 等人调查了人们对三个公平定义的看法，大约 200 名来自亚马逊 “土耳其机器人” (Mechanical Turk) 的参与者被要求在两名申请贷款的个人身上选择他们对三种分配规则的偏好。结果表明，有一个明确的偏好分配资源的方式，按申请人贷款偿还率分配资源。

6. Support for Fairness Improvement. Metevier 等人 (NIPS 2019) 提出了 RobinHood，该算法支持离线上下文赌博机场景下的多个自定义公平性定义。RobinHood 利用 concentration inequalities 来计算高概率界，并寻找满足公平性要求的解。当需求被违反时，它会给用户警告。该方法在三个应用场景下进行了评估，实验证明了 RobinHood 算法优于其他算法。

   Albarghouthi 和 Vinitsky 提出了 “fairness-aware programming” 的概念，其中公平性是首要关注的问题。为了帮助开发人员定义他们自己的公平性规范，他们开发了一种规范语言。与传统测试中的断言一样，公平性规范被开发成运行时监视代码，以捕获违规行为。

6.5.2 Test Generation Techniques for Fairness Testing

本节描述的是测试生成技术中的公平性测试。Galhotra 等人提出了利用因果分析考虑群体公平的 Themis。它将公平分数定义为公平的衡量标准，并使用随机测试生成技术来评估歧视程度。据报道，Themis 在更多歧视的系统上效率更高。

Themis 为群体公平随机生成测试样例，而 Udeshi 等人提出了 Aequitas，侧重于测试生成，以揭示歧视性输入和那些对理解个人公平至关重要的输入。生成方法首先对输入空间进行随机抽样以发现是否存在歧视输入，然后搜索这些输入的邻域以找到更多的输入。除了检测公平漏洞，Aeqitas 还对机器学习模型进行了再训练，减少了这些模型决策中的歧视。

Tramer 等人 (ICSEM 2017) 最先提出了公平漏洞的概念。他们认为在受保护的属性和算法输出之间有统计学意义的关联是一个公平漏洞，命名为 unexplained Associations。他们提出了第一个全面的测试工具，旨在帮助开发人员测试和调试公平漏洞，并提供易于理解的漏洞报告。

6.6 Interpretability

大概看了一下，觉得这部分有点扯淡这里不做过多的描述，有兴趣的同学自己看看。

6.7 Privacy

Ding 等人将程序视为灰盒，并通过统计测试检测不同的隐私侵犯。对于检测到的违规情况，他们生成反例来说明这些违规，并帮助开发人员理解和修复错误。

7 ML TESTING COMPONENTS

本节通过确定 ML 测试可能揭示出现错误的组件 (数据、学习程序或框架) 来组织 ML 测试工作。

7.1 Bug Detection in Data

数据是 ML 测试中需要测试的组件，因为 ML 系统的性能很大程度上依赖于数据。Breck 等人所指出的，应该尽可能早的发现数据漏洞，因为来自训练模型的预测经常被记录下来，并用于生成进一步的数据。这最后的迭代创造了一个反馈循环，可能会随着时间的推移放大开始很小的数据漏洞。

然而，数据测试非常具有挑战性。根据 Amershi et al. 的研究，在微软开发 AI 应用程序时，数据的管理和评估是最具挑战性的任务之一。Breck 等人提到，数据生成逻辑通常在 ML pipline 中缺乏可见性。

7.1.1 Bug Detection in Training Data

Debugging Framework Development.

Hynes 等人提出了一种 ML 工具 data linter，用于自动检查 ML 数据集。他们考虑了三种类型的数据问题:1) 数据编码错误，如将数字或日期误输入字符串;2) 异常值和缩放，如不常见的列表长度;3) 包装错误，如重复的值，空的例子，和其他数据组织问题。Cheng 等人提出了一系列指标来评估训练数据是否涵盖了所有重要的场景。

Performance-based Data Bug Detection. 为了解决训练数据中的问题，Ma 等人 (FSE 2018) 提出了 MODE。MODE 识别出导致分类错误的神经网络中的故障神经元，并通过数据重采样对训练数据进行测试，分析故障神经元是否受到影响。通过在各种数据集进行评估，MODE 将测试效率平均从 75% 提高到 93%。

7.1.2 Bug Detection in Test Data

Metzen 等人 (ICLR 2017) 提出用一个小的子网络来增强 dnn，专门用于区分真实数据和包含敌对扰动的数据。Wang 等人 (ICSE 2019) 通过观察到对抗样本对扰动更敏感，利用 DNN 模型突变暴露了对抗样本。评估基于 MNIST 和 CIFAR10 数据集。该方法检测到 96.4% / 90.6% 的 MNIST/CIFAR10 突变的对抗样本 74.1/86.1。

7.1.3 Skew Detection in Training and Test Data

Breck 调查了训练数据和服务数据 (ML 模型部署后预测的数据) 中的偏差。为了检测特征中的倾斜，它们进行键连接特征比较。为了量化分布中的偏差，他们认为一般的方法，如 KL 散度或余弦相似度，对于软件开发者来说可能不够直观。相反，他们建议使用概率的最大变化值作为两个分布中的一个值来度量它们的距离。

7.1.4 Frameworks in Detecting Data Bugs

Breck 等人 (SysML, 2019) 提出了一种用于检测数据 bug 的数据验证系统。该系统应用约束条件 (例如类型、域) 在单个 batch(训练数据或新数据内) 中寻找错误，并量化训练数据和新数据之间的距离。他们的系统被部署为 TFX(谷歌的端到端机器学习平台) 的一个组成部分。实验证明了该系统有助于及早发现和调试数据 bug。他们还总结了数据错误的类型，其中 new feature column, unexpected string values, 和 missing feature columns 是最常见的三种错误。Krishnan 等人 (VLDB 2016) 提出了一个模型训练框架 ActiveClean，该框架迭代地数据清理，同时可证明其收敛性。ActiveClean 根据数据的值和数据是脏数据的可能性，给开发者一个建议清理的数据样本清单。然后，分析人员可以对数据应用值转换和过滤操作，以清除脏数据。

2017 年，Krishnan 等人提出了一个名为 BoostClean 的系统，用于检测训练数据中的域值违规 (即属性值在允许的域之外)。该系统使用 Isolate Forest 算法用可用的清理资源来改进模型的性能。该系统在解决了检测到的问题后，能够提高预测精度，比最佳非集成方案提高 9%。

ActiveClean 和 BoostClean 在测试过程中，都涉及到人的操作。Schelter 等人关注于大规模数据集中的自动单元检测。

8 APPLICATION SCENARIOS

这里只关注于自动驾驶这一种应用场景，机器翻译和自然语言推断就不做过多的思考了。

自动驾驶汽车测试的历史相对较长。正如 Woehrle 等人指出并讨论的那样，自动驾驶汽车测试也有许多研究机会和开放性问题。最近，基于搜索的 AV 测试生成已经成功应用。Abdessalem 等 (ICSE 2018) 致力于提高自动驾驶汽车高级驾驶员辅助系统 (ADAS) 基于搜索的测试效率和准确性。他们的算法使用分类模型来提高关键场景下基于搜索的测试生成的效率，进一步使用搜索算法细化分类模型，

以提高其准确性。Abdessalem 等人提出了一种多目标搜索算法 FITEST，该算法用来搜索违反系统需求或导致失败的特征交互。

目前市场上投入使用的大多数自动驾驶汽车系统都是半自动驾驶汽车，需要人类驾驶员作为备用。导致人类驾驶员控制车辆的问题被称为 Disengagement。

Banerjee 等人从 12 家自动驾驶汽车制造商的 144 辆汽车的数据中调查了 5328 次脱离驾驶的原因和影响，这些汽车累计行驶 1,116,605 英里，其中有 42 次 (0.8%) 导致了事故。他们将脱离的原因分为 10 种类型。64% 的脱离是由机器学习系统中的错误引起的，其中图像分类行为 (例如，交通信号灯、车道标记、洞和颠簸) 是 44%。剩下的 20% 是由于控制和决策框架中的错误，如不恰当的运动规划。

Pei 等人使用基于梯度的差分测试生成测试输入来检测潜在的 DNN bug，并利用神经元覆盖性作为指导。Zhang 等人提出了 DeepRoad，一种基于 GAN 的生成真实驾驶场景测试图像的方法。他们的方法能够支持两种天气条件的测试 (即下雪和下雨)。图片是用 YouTube 视频中的图片生成的。Zhou 等人提出了 DeepBillboard，它可以生成真实世界的 DeepBillBoard，从而触发自动驾驶系统的潜在转向错误。此工作提供了实际的自动驾驶系统生成连续和真实的物理世界测试的可能性。

Wicker 等人使用特征引导的蒙特卡洛树搜索 (Monte Carlo Tree Search) 来识别图像中最容易影响自动驾驶系统的元素————对抗样本。Uesato 等人的目标是在强化学习中发现导致 agent(如自动驾驶) 发生灾难性故障的关键因素。他们论证了传统随机测试的局限性，然后提出了一种可预测的对抗样本生成方法来预测失败，估计可靠风险。对 TORCS 模拟器的仿真结果表明，所提出的方法在较少的 Monte Carlo 运行情况下是有效的。

为了测试算法是否会导致有问题的模型，Dreossi 等人提出了生成训练数据和测试数据的方法。他们利用卷积神经网络 (CNN)，构建了生成自然图像的图像，并将收集到的信息可视化，以检测自动驾驶场景下的盲点或角落情况。

O Kelly 等人 (NIPS 2018) 提出了一种基于风险的自动驾驶汽车测试框架，以预测交通行为的基本分布中发生事故的概率。他们认为形式化的验证 AV 系统的正确性是不可行的。传统测试在真实环境中测试自动驾驶汽车需要大量的时间。为了解决这些问题，他们将 AV 测试视为专门对罕见事件的模拟，然后通过评估事故概率来加速 AV 测试过程。

9 ANALYSIS OF LITERATURE REVIEW

本节分析了不同测试属性和机器学习类别之间的研究分布。

9.1 Timeline

下图中展示了 ML 测试开发中的几个关键贡献。早在 2007 年，Murphy 等人就提到了测试机器学习应用的想法。考虑到获得测试 oracle 的难度，他们将机器学习应用归类为“不可测试”的程序。主要考虑实现 bug 的检测，确保应用程序的正确实现，并满足了用户的期望。后来，Murphy 等人讨论了机器学习算法的特性，这些特性可以作为 Metamorphies Relations 来检测实现错误。

2009 年，Xie 等人也在监督学习应用中应用了 Metamorphies Testing。公平性检验由 Dwork 等人在 2012 年提出；可解释性问题是 Burrell 在 2016 年提出的。

2017 年，Pei 等人发表了第一篇关于深度学习系统的白盒测试论文。他们的工作率先提出了 DNN 的覆盖标准。在本文的启发下，出现了一些机器学习测试技术，如 DeepTest、DeepGauge、DeepConcolic、

DeepRoad 等。许多软件测试技术已经应用于 ML 测试，如不同的测试覆盖标准、变异测试、组合测试、变形测试和模糊测试。

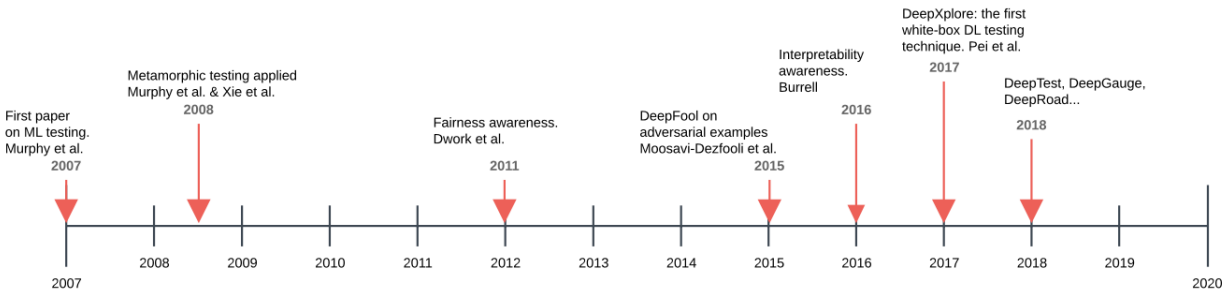

图 4: Machine Learning testing Timelines

9.2 Research Distribution among Machine Learning Categories

9.2.1 Research Distribution between General Machine Learning and Deep Learning

为了方便区分，将 ML testing 分成两类，只针对 deep learning 的测试，通用的 machine learning 的测试。**2017 年之前，论文主要集中在通用机器学习测试方面；在 2018 年之后，一般机器学习和深度学习的特定测试都显著增加**。

9.2.2 Research Distribution among Supervised/Unsupervised/Reinforcement Learning Testing

研究监督学习的文章非常多，而研究无监督学习和强化学习的文章非常的少。我比较关注的是强化学习领域，Uesato 等人 (ICLR 2019) 提出了一种预测对抗样本生成方法，用于预测强化学习中的失败和估计可靠风险。所以，目前 RL 算法的 testing 依然有非常多的机会和挑战。

9.3 Research Distribution among Different Testing Properties

目前的有关 Testing Properties 的研究分布也非常的不平衡。有关正确性和鲁棒性的研究较多，而其他的研究较少。

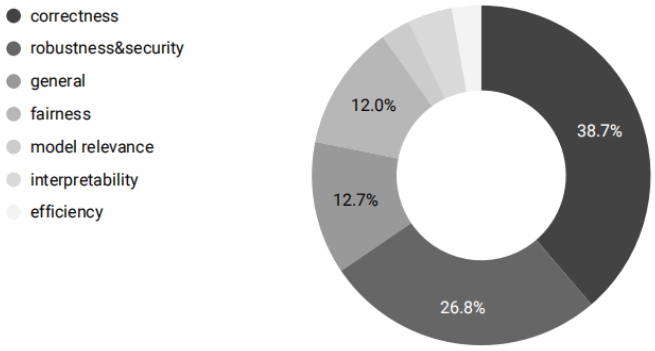

图 5: Research distribution among different testing Properties

9.4 Open-source Tool Support in ML Testing

目前，有几个专门为 ML 测试设计的工具。angel 等人提出了 Themis，一个测试群体歧视的开源工具。还有一个用于 tensorflow 的 ML 测试框架，名为 mltest，用于编写简单的 ML 单元测试。与 mltest 类似，有一个用于为基于 pytorch 的 ML 系统编写单元测试的测试框架，名为 torchtest16。Dolby 等扩展了 WALA，使用 TensorFlow 对机器学习代码进行静态分析。与传统的测试相比，现有的工具对 ML 测试的支持相对不成熟。对 ML 测试的工具支持改进仍然有很大的空间。

10 CHALLENGES AND OPPORTUNITIES

10.1 Challenges in ML Testing

10.1.1 Challenges in Test Input Generation

测试输入生成仍然具有挑战性，因为 ML 模型的行为空间很大。基于搜索的软件测试生成 (SBST) 使用元启发式优化搜索技术，如遗传算法，自动生成测试输入。它是一种测试生成技术，在传统软件测试范式的研究中得到了广泛的应用。除了生成用于测试功能属性 (如程序正确性) 的测试输入外，SBST 还被用于探索需求分析中算法公平性的紧张关系。SBST 已成功应用于自动驾驶系统的测试 (ICSE 2016)。由于 SBST 和 ML 之间明显的可以相互使用，因此有许多研究机会将 SBST 应用于生成用于测试其他 ML 系统的测试输入；SBST 自适应地在大输入空间中搜索测试输入。

现有的测试输入生成技术侧重于生成对抗性输入来测试 ML 系统的鲁棒性。然而，对抗性样本经常受到争议，因为它们不代表真正的输入数据。因此，有一个有趣的研究方向是如何生成自然测试输入，以及如何自动测量生成输入的自然度。

已经有工作试图生成测试输入尽可能在自然场景下的自主驾驶，如 DeepTest，DeepHunter 和 DeepRoad，但仍可能生成不自然的图像：有时人类可能不承认这些工具生成的图像。探索这些看上去对人类没有意义的测试数据是否应该被采用/有效用于 ML 测试是一件既有趣又具有挑战性的事情。

10.1.2 Challenges on Test Assessment Criteria

有很多评估生成的测试样例质量和充足性的文章。然而，对于不同的评估指标之间如何相互关联，或者这些评估指标如何关联测试故障检测能力，仍然缺乏系统的评估。测试评价标准与测试充分性之间的关系尚不明确。此外，评估标准可能提供一种解释和理解 ML 模型行为的方法，这可能是进一步探索的一个有趣的方向。

10.1.3 Challenges Relating to The Oracle Problem

Metamorphic relations 是有效的 oracles，但在大多数情况下，它们需要由人类的聪明才智来定义。因此，剩下的一个挑战是自动识别和构造可靠的测试 oracle 用于 ML 测试

Murphy 等人讨论了当涉及浮点计算时，蜕变测试中可能出现的 flaky(不靠谱) 测试。Flaky 测试检测是传统软件测试中一个具有挑战性的问题。由于 oracle 问题，在 ML 测试中可能更具有挑战性。

即使没有不靠谱的测试，伪预言也可能不准确，导致许多假阳性。因此，有必要探索如何产生更准确的测试预言，以及减少报告问题中的假阳性。我们甚至可以使用 ML 算法 b，来学习在测试 ML 算法 a 中检测到的假阳性 oracle。

10.1.4 Challenges in Testing Cost Reduction

在传统的软件测试中，成本问题仍然是一个大问题。目前，有很多降低成本的技术，如测试选择、测试优先级划分和测试执行结果预测。在 ML 测试中，成本问题可能更严重，特别是在测试 ML 组件时，因为 ML 组件测试通常需要模型再训练或重复预测过程。它也可能需要数据生成来探索巨大的行为空间。

降低成本的一个可能的研究方向是将 ML 模型表示为一种中间状态，以便于测试。还可以应用传统的成本降低技术，如测试优先级或最小化，以在不影响测试正确性的情况下减少测试用例的规模。

更多的 ML 解决方案被部署到不同的设备和平台 (例如，移动设备，物联网边缘设备)。由于目标设备的资源限制，如何在不同的设备上有效地测试 ML 模型以及部署过程也是一个挑战。

10.2 Research Opportunities in ML testing

10.2.1 Testing More Application Scenarios

目前的研究主要集中在监督学习上，特别是分类问题。**关于测试无监督和强化学习的相关问题还需要更多的研究**。目前文献中的测试任务主要集中在图像分类方面。在许多其他领域，如语音识别、自然语言处理和智能体玩游戏，仍有很多的测试研究机会。

10.2.2 Testing More ML Categories and Tasks

测试更多的 ML 类别和任务。不同机器学习类别和 30 个任务的测试技术的覆盖明显不平衡。测试无监督和强化学习系统既有挑战，也有研究机会。例如，迁移学习是一个最近很受关注的话题，它侧重于存储在解决一个问题时获得的知识，并将其应用于另一个不同但相关的问题。迁移学习测试也很重要，但在现有的文献中涉及面很窄。

10.2.3 Testing Other Properties

我们可以看到，大多数工作测试鲁棒性和正确性，而相对较少的论文 (少于 3%) 研究效率，模型相关性，或可解释性。模型相关性检验具有挑战性，因为未来数据的分布往往是未知的，而许多模型的能力也是未知的，难以衡量。

对 ML 模型中模型相关性差的发生率，和模型相关性差与高安全风险之间的平衡进行实证研究非常的重要。对于测试效率，需要在不同的层次上测试效率，比如在不同平台、机器学习框架和硬件设备之间切换时的效率。为了测试属性的可解释性，现有的方法主要依赖于人工评估，它检查人类是否能够理解 ML 模型的逻辑或预测结果。研究可解释性的自动评估和可解释性违规的检测也是很有趣的。

对于公平性和可解释性的定义和理解缺乏共识。因此，需要在不同的背景下进行更明确的定义、形式化和实证研究。有一种讨论认为，机器学习测试和传统软件测试在保证不同属性方面可能有不同的要求。因此，需要更多的工作来探索和识别那些对机器学习系统最重要的特性。

10.2.4 Presenting More Testing Benchmarks

据我们所知，像 CleverHans 这样专门为 ML 测试研究目的而设计的基准测试非常少，比如对抗性示例构建。所以，需要更多专门为 ML 测试设计的基准测试。例如，一个带有真正错误的机器学习程序库可以为错误修复技术提供一个很好的基准。

10.2.5 Covering More Testing Activities

一个好的需求分析可以处理许多非功能属性，比如公平性。现有的工作集中于离线测试。在线测试需要更多的关注。根据 Amershi 等人的研究，数据测试尤其重要。

此外，在 ML 测试中还有很多回归测试、bug 报告分析和 bug 分类的研究机会。由于机器学习算法的黑盒特性，与传统的软件测试相比，ML 测试结果往往更难让开发人员理解。在 ML 测试中，测试结果的可视化可能特别有帮助，可以帮助开发人员理解 bug，并帮助进行 bug 本地化和修复。

10.2.6 Mutating Investigation in Machine Learning System

目前，有一些研究讨论了突变的机器学习代码 (QRS 2018)，但没有研究探索如何更好地为机器学习代码设计突变操作符，从而使突变体能够更好地模拟真实世界的机器学习系统的 bug。