arXiv:2301.03837v1 [cs.SE] 10 Jan 2023

理解机器学习赋能系统的复杂性及其对测试的影响

Rasa 案例研究

Junming Cao*、Bihuan Chen*、Longjie Hu*、Jie Gao† Kaifeng Huang* 和 Xin Peng*
*中国上海复旦大学
†新加坡新加坡科技设计大学

摘要——随着机器学习领域近期取得突破，机器学习赋能系统不断涌现。文献中普遍采用以模型为中心的视角，只关注机器学习模型的分析。然而，只有少数研究从系统视角出发，探讨机器学习组件如何与系统协同工作，以及它们如何影响机器学习赋能系统的软件工程。本文采用这一系统视角，对 Rasa 3.0 开展案例研究。Rasa 3.0 是一款工业级对话系统，已被全球众多公司广泛采用。我们的目标是刻画此类大规模机器学习赋能系统的复杂性，并了解这种复杂性对测试的影响。研究揭示了机器学习赋能系统软件工程方面的实际启示。

一、引言

近年来，机器学习（ML）取得的进展引发了人们日益浓厚的兴趣，促使各行各业纷纷应用机器学习，例如自动驾驶汽车、虚拟助手、机器人技术和医疗保健。根据 IBM 发布的全球人工智能采用指数 [34], 35，全球有 % 的公司已在业务中部署人工智能，另有 42% 的公司正在探索人工智能。由此趋势催生了机器学习赋能系统，这类系统由机器学习组件和非机器学习组件构成。机器学习组件往往至关重要，但通常只是机器学习赋能系统众多组件中的一部分 [38]。

以往针对机器学习软件工程的研究往往采取以模型为中心的视角，只关注机器学习模型的分析 [38, 55]。例如，深度学习模型测试（如 [4, 19, 25,40, 59, 68, 72, 82, 84]）、验证（如 [9, 57, 58, 66, 73]）和调试（如 [43, 49, 54, 71]）方面均已取得诸多进展。只有少量研究从整体系统视角出发，例如探讨架构设计[64, 80], 、技术债务 [63, 70], 、机器学习组件纠缠[53, 78, 83], 、特征交互 [1, 2, 8], 以及 Apollo 中的模型交互 [60]。然而，对机器学习赋能系统缺乏系统级认识，可能会掩盖工程实践中的问题，并阻碍切实可行的解决方案。

本文采用这种系统视角，并对 Rasa 3.0 [11] 开展案例研究，以刻画此类大规模机器学习赋能系统的复杂性，并了解复杂性对测试的影响。Rasa 是一个面向任务的工业级对话系统，已被全球各地的众多公司广泛使用。因此，我们认为 Rasa 能够很好地代表现实世界中的机器学习赋能系统。

我们首先从三个层面考察 Rasa 的复杂性。在系统层面，我们探究机器学习组件如何应用于 Rasa 的各个模块。我们发现，6 个模块中共有 15 个机器学习组件，包含 23 个机器学习模型。在交互层面，我们分析机器学习组件如何与 Rasa 中的其他组件交互。我们发现，4 个主要类别和 8 个子类别中共有 43 种交互模式和 230 个交互实例。在组件层面，我们探究机器学习组件的代码由哪些类型的代码组成。我们发现，组件内部 57.1% 的代码是数据处理代码，数据处理代码与模型使用代码之间存在 8 种组合模式。

随后，我们从两个角度探讨复杂性对测试的影响。从测试实践的角度，我们分析测试用例的特征，以及它们应对复杂性的能力。我们发现，由于庞大的配置空间以及隐藏的组件交互所带来的复杂性，组件交互的测试覆盖率较低。从变异测试的角度，我们研究测试用例和测试数据（即用于测试模型的数据）的缺陷发现能力，以及它们应对复杂性的能力。我们发现，由于数据处理代码带来的复杂性，数据处理代码中可能存在许多只能通过测试用例检测到的潜在缺陷。由于庞大的配置空间带来的复杂性，测试数据杀死变异体的能力有限。

基于我们的案例研究，我们强调了改进机器学习赋能系统软件工程的实践启示。例如，应充分测试机器学习赋能系统的配置空间，并向开发者提供配置建议。应构建通用的数据处理代码分类体系，进而开发相应的维护和测试工具。应创建更多集成级测试用例，以覆盖组件交互。应结合使用测试用例和测试数据，以检测非机器学习特定缺陷和机器学习特定缺陷。

总而言之，本文作出了以下贡献。

- 我们对 Rasa 开展了深入的案例研究，以刻画其复杂性及其复杂性对测试的影响。
- 我们重点阐述了可用于改进机器学习赋能系统软件工程的实践启示。

II. 背景与研究设计

我们介绍了典型面向任务的对话系统的架构、Rasa 概述以及我们的研究设计。

A. 典型面向任务的对话系统架构

面向任务的对话系统（TDS）旨在帮助用户完成特定任务，例如餐厅预订和航班预订 [15]。基于流水线的面向任务对话系统由四个部分组成，即自然语言理解（NLU）、对话状态跟踪（DST）、对话策略（Policy）和自然语言生成（NLG） [86]。自然语言理解将用户话语解析为结构化的语义表示，其中包括意图和槽值。意图是对用户话语进行的高层分类，例如告知 和 请求。槽值是用户话语中提及的、特定于任务的实体，例如餐厅价格范围和位置偏好。对用户话语进行分词和特征化后，自然语言理解会应用分类模型来识别意图，并应用命名实体抽取模型来提取槽值。对话状态跟踪会综合整个对话历史，包括带有预测意图和槽值的用户话语以及系统响应，以估计当前对话状态；这通常被表述为序列预测任务[77]。对话状态通常是截至当前时间戳的用户意图和槽值的概率分布。根据估计出的对话状态，策略会生成下一步系统动作，例如 查询数据库 和 提出问题。由于策略会依次确定一系列动作，因此会采用循环神经网络（RNN）等序列模型。对于需要作出响应的动作，自然语言生成会将动作转换为自然语言话语，这通常被视为序列生成任务 [76]。

B. Rasa 3.0 概述

Rasa 是一种广受欢迎的开源机器学习赋能的面向任务对话系统，完全使用 Python 实现，被许多知名公司用于面向真实用户的客户服务，其中包括 Adobe、空中客车和 N 26 [11]。Rasa 3.0 的架构概览如图所示。1。每个模块由一个或多个语义相似的组件组成。除了典型对话系统中的模块外，Rasa 还提出了选择器模块，用于为常见问题选择候选意图和回复 [16]。为便于阐述，我们介绍 Rasa 3.0 中的一些概念。

Rasa 中的组件。Rasa 中有两种类型的组件。我们将由机器学习模型实现的组件定义为机器学习组件，将通过基于规则的代码逻辑实现的组件定义为基于规则的组件。本文不考虑 Rasa 中的一般实用代码，例如命令行和数据库访问代码。

Rasa 中的配置文件和组件图。由于每个模块中都有多个可用组件，开发人员需要通过配置文件 选择实际用于 Rasa 流水线的组件，以构建聊天机器人。每个组件的参数都在配置文件中指定（例如，组件使用的机器学习模型以及机器学习模型的超参数）。Rasa 使用 Dask [6] 将配置文件编译为组件图。组件图中的每个节点表示一个组件，与其相连的边表示与之存在输入和输出数据依赖的上游和下游组件。组件的执行遵循边所指定的拓扑顺序。这些组件通过共享的 Message 类实例中的字段彼此交互。上游组件将输出存储到 Message 实例中，下游组件则从中获取输出以进行进一步处理。

Rasa 中的机器学习阶段。与 Apollo 不同，Apollo 使用来自外部系统的训练好的模型文件，因此只包含机器学习模型的推理阶段 [60]，；Rasa 则包含机器学习模型的训练、评估和推理阶段。给定配置文件后，Rasa 会分别将其编译为训练组件图和推理组件图。在训练阶段，首先训练可训练的上游组件，然后处理下游组件使用的训练数据。在评估阶段，仅报告意图分类器、实体提取器和策略的性能指标，因为其他模块的评估数据没有真实标注。

C. 研究设计

我们的目标是了解 Rasa 中的复杂性及其对测试的影响。为实现这一目标，我们提出了五个研究问题。

- **研究问题1 系统复杂度分析**：机器学习组件如何在 Rasa 的各个模块中采用？
- **研究问题2 交互复杂度分析**：机器学习组件如何与 Rasa 中的其他组件交互？
- **RQ3 组件复杂度分析**：机器学习组件的代码由哪些类型的代码组成？
- **RQ4 测试实践分析**：测试用例具有什么特征？它们应对复杂性的效果如何？
- **RQ5 变异测试分析**：测试用例和测试数据（即用于测试模型的数据）的缺陷发现能力如何？它们应对复杂性的能力又如何？

研究问题1 旨在识别 Rasa 中的机器学习组件，并从依赖库和机器学习模型的角度对其进行总体考察。**研究问题2** 旨在总结一套全面的组件交互模式分类体系。**RQ3** 旨在检查每个组件内部的源代码，以描述不同代码类型（包括数据处理代码、模型使用代码等）的统计特征和组合模式。我们从**研究问题1**、**研究问题2** 和**RQ3** 中得出的发现，能够揭示复杂性在现实世界大型机器学习赋能系统中的起源及表现，为从业者和研究人员提供洞见，帮助他们克服实现、维护、调试和测试此类复杂系统所涉及的种种复杂性。

RQ4 旨在从代码覆盖率、测试用例统计（即粒度级别、预言机类型和机器学习阶段）以及组件交互覆盖率等角度，对 Rasa 的测试用例进行定量评估。**RQ5** 旨在生成变异体（即人工注入的缺陷），并检查测试用例能否杀死（即检测到）这些变异体。此外，对于存活的变异体，我们使用 3 个默认配置文件在 *MutiWoz* [15], 这一广泛使用的多领域对话系统数据集上训练 Rasa。我们对比变异后的 Rasa 代码所得到的性能指标与使用无缺陷代码训练的流水线所得到的指标，并计算其统计显著性。**RQ4** 和**RQ5** 的研究结果评估了 Rasa 中的测试实践，并

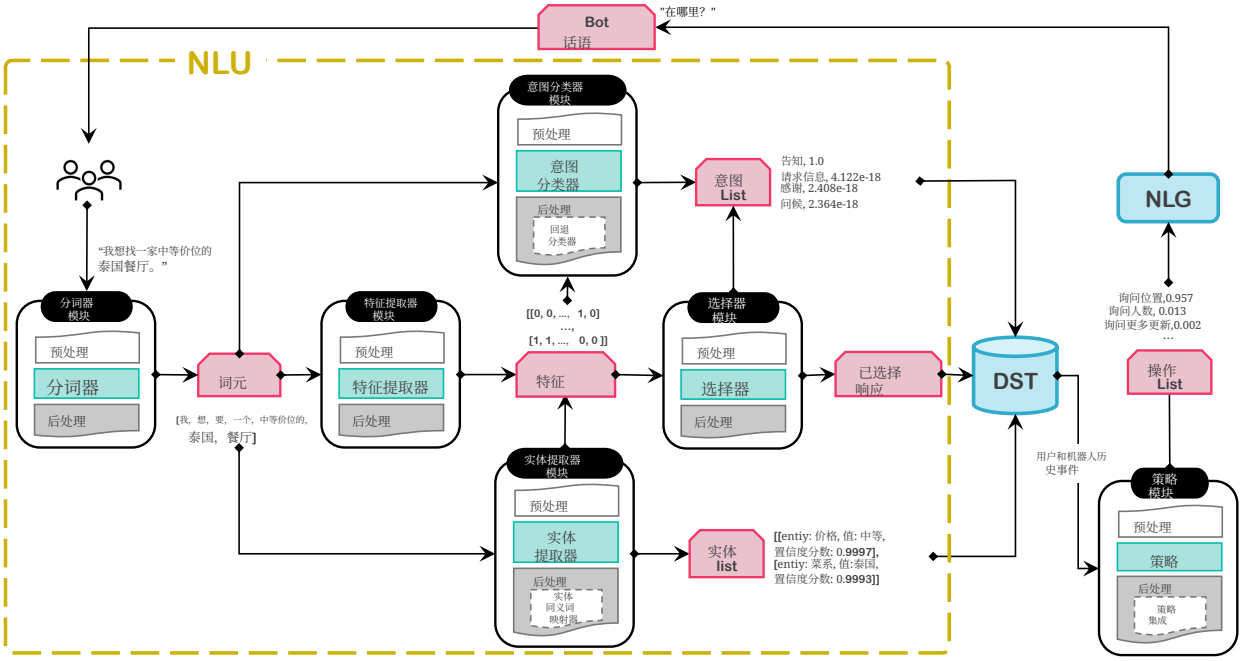


图1：Rasa 的模块和工作流

为复杂的机器学习赋能系统中的自动化测试生成、缺陷定位和错误修复技术提供启示。

III. 研究问题1：系统复杂性分析

A. 方法

为回答**研究问题1**，我们识别了 Rasa 中的机器学习组件和基于规则的组件，并通过详细检查 Rasa 的源代码和文档对其进行了刻画。由于对话状态跟踪和自然语言生成在 Rasa 中完全通过基于规则的代码逻辑实现，且不包含机器学习组件，因此我们将其排除在外。

我们识别出的所有模块均列于表 I 中，但一个特殊模块除外，即 共享，因为它包含通用的数据处理代码和机器学习模型定义代码（例如 Transformer），但不包含任何独立组件。我们将在后面三个研究问题中将其纳入分析。具体而言，对于每个组件，我们递归跟踪其中的方法，以手动提取模型或规则定义代码。我们通过阅读外部库的文档和源代码，检查了机器学习库中应用程序接口的实现细节，包括机器学习模型类型和候选模型数量。

具体而言，我们分析了机器学习组件是通过外部直接库还是间接库实现的，这些组件是否可以训练（请注意，不仅某些机器学习组件可以训练，只要某些基于规则的组件在处理训练数据时会更新内部参数，它们也可以训练），Rasa 是否使用自有代码实现组件并提供内置的模型和规则定义代码，以及每个组件的代码行数（LoC；不包括空行、代码注释和导入语句）。

B. 结果

结果汇总于表I中。灰色显示的组件为机器学习组件，其余为基于规则的组件。总计有 6 个模块，包括 15 个机器学习组件和 14 个基于规则的组件。这些组件包含 23 个机器学习模型，其实现依赖于 7 个直接依赖的外部机器学习库和 3 个间接依赖的外部机器学习库。特别是，分词器 和特征提取器中的所有机器学习组件均不可训练，因为其中使用了预训练语言模型。策略 中的所有组件均使用 Rasa 自有代码实现，因为现有库并未提供可直接使用的策略模型。机器学习组件共有 5348 行代码，基于规则的组件共有 2980 行代码。此外，通用模块共享 包含 5375 行代码，未列入表中。

值得注意的是，我们发现，经典机器学习模型（例如支持向量机和条件随机场）与深度学习模型（例如卷积神经网络和 Transformer）在 Rasa 中发挥着重要作用。这与此前关于 Apollo 的研究 [60] 不同，后者侧重于深度学习模型。

N接下来，我们介绍各个模 dule.

分词器。 分词器使用特定组件的分隔符（例如空格和标点符号）将用户话语拆分为词元。(1) *SpacyTokenizer* 提供最丰富的词元信息，包括依据规则拆分词元、使用查找表进行词元词形还原，以及使用多层感知机（MLP）进行词性标注。(2) *JiebaTokenizer*是唯一使用隐马尔可夫模型（HMM）对非英语句子进行分词的组件 [20]。(3) *MitieTokenizer*和 *WhitespaceTokenizer* 根据预定义规则对文本进行分词。

特征提取器。如图 1 所示，特征提取器将词元转换为特征，以供下游模块推理。(1) *ConveRT* 特征提取器 加载 TFHub 的 [27]预训练 ConveRT（Transformer 对话表示）TensorFlow 模型来提取词元特征 [29]。(2) 语言模型特征提取器 加载

表 I：Rasa 的系统复杂性分析

模块	组件	直接依赖库	间接依赖库	模型类型	模型数量	可训练	Rasa 实现	LoC
分词器	JiebaTokenizer	结巴	N/A	HMM	1	否	否	85
	SpacyTokenizer	spaCy	Thinc	MLP	1	否	否	39
	Mitie 分词器	Mitie	N/A	N/A	N/A	否	否	43
	WhitespaceTokenizer	N/A	N/A	N/A	N/A	False	True	52
特征提取器	ConveRT特征提取器	TensorFlow	N/A	Transformer	1	False	否	269
	语言模型特征提取器	Transformer	TensorFlow	Transformer	6	否	假	378
	Mitie 特征提取器	Mitie	Dlib	CCA	1	假	假	98
	spaCy 特征提取器	spaCy	Thinc	CNN	2	否	否	66
	计数向量特征提取器	Scikit-learn	N/A	N/A	N/A	True	错误	520
	词汇句法特征提取器	N/A	N/A	N/A	N/A	True	True	319
	正则表达式特征提取器	N/A	N/A	N/A	N/A	True	True	151
意图分类器	DIET 分类器	TensorFlow	N/A	Transformer	1	True	True	1217
	Mitie 意图分类器	Mitie	Dlib	SVM	1	True	False	89
	Scikit-learn 意图分类器	Scikit-learn	N/A	SVM	1	True	False	173
	回退分类器	N/A	N/A	N/A	N/A	False	True	91
	关键词意图分类器	N/A	N/A	N/A	N/A	True	True	132
实体提取器	DIET 分类器	TensorFlow	N/A	Transformer	1	True	True	1217
	CRF 实体提取器	Scikit-learn	N/A	CRF	1	True	假	438
	Mitie 实体提取器	Mitie	Dlib	SVM	1	True	False	164
	spaCy 实体提取器	spaCy	Thinc	MLP	1	False	否	52
	Duckling 实体提取器	N/A	N/A	N/A	N/A	否	否	134
	正则表达式实体提取器	N/A	N/A	N/A	N/A	True	True	124
	实体同义词映射器	N/A	N/A	N/A	N/A	True	True	102
选择器	响应选择器	TensorFlow	N/A	Transformer	2	True	True	560
策略	TED策略	TensorFlow	N/A	Transformer+CRF	1	True	True	1262
	UnexpecTED 意图策略	TensorFlow	N/A	Transformer+CRF	1	True	True	458
	记忆化策略	N/A	N/A	N/A	N/A	True	True	207
	增强记忆化策略	N/A	N/A	N/A	N/A	True	True	65
	规则策略	N/A	N/A	N/A	N/A	True	True	818
	策略集成	N/A	N/A	N/A	N/A	False	True	150

Hugging Face Transformers 中的预训练语言模型[22], , 包括 BERT [17], GPT [31], XLNet [79], RoBERTa[46], XLM [41] 和 GPT2 [56]。(3) *Mitie* 特征提取器 将典型相关分析（ CCA ）特征与词语形态特征结合起来。(4) *spaCy* 特征提取器 根据配置文件中指定的预训练 spaCy 流水线，应用 HashEmbedCNN 或 RoBERTa 将词元转换为特征。(5) 计数向量特征提取器、词汇句法特征提取器 和正则表达式特征提取器 分别使用 n 元语法、滑动窗口和正则表达式模式创建稀疏特征。

意图分类器。意图分类器根据上游模块提供的词元和特征，生成按置信度分数排序的预测意图列表。(1)*DIET* 分类器通过双重意图与实体变换器（DIET）同时执行意图分类和实体识别，因此同时属于**意图分类器**和**实体提取器**模块。(2)*Mitie* 意图分类器和*Scikit-learn* 意图分类器分别使用 Scikit-learn 和 Mitie，通过稀疏线性核 [65] 应用多类支持向量机（SVM）。(3)关键词意图分类器使用从训练数据中提取的关键词对用户意图进行分类。(4)回退分类器是一个后处理组件，用于检查*DIET* 分类器中其他组件的结果。如果置信度分数未超过阈值，或者排名最高的两个意图的分数差小于歧义_阈值，它会将用户话语识别为意图 nlu_fallback。

实体提取器。实体提取器会提取诸如餐厅位置和价格等实体。(1) *DIET* 分类器 也可用作实体提取器。(2) *CRF* 实体提取器、*Mitie* 实体提取器 和 *spaCy* 实体提取器 分别使用条件随机场（CRF）模型、多类线性支持向量机和多层感知机来预测实体。(3) *Duckling* 实体提取器 和正则表达式实体提取器 使用 Duckling 服务器提取实体[23] 和正则表达式模式。(4) 实体同义词映射器 是一个后处理组件，用于将同义实体值转换为相同的值。如图1 所示，“价格”实体的值“moderate”会被实体同义词映射器转换为“mid”。

选择器。响应选择器旨在直接从一组候选响应中选出响应，这在相关文献中也称为响应选择任务 [16]。它将用户输入和候选响应嵌入同一个向量空间，并采用与*DIET* 分类器相同的神经网络架构。

策略。 策略根据对话状态决定系统在每次对话中采取的动作。(1) *TED*策略 提出了一种 Transformer 嵌入式对话（TED）模型，将对话状态和系统动作嵌入到单一的语义向量空间中，并选择与当前对话状态相似度得分最高的动作 [75]。(2) 记忆化策略、增强型记忆化策略 和规则策略 将当前对话历史与训练数据中的示例及预定义规则进行匹配，以预测系统动作。(3)意外意图策略根据当前对话状态决定意图分类器所预测意图的可能性，其模型架构与 *TED*策略相同。(4) 策略集成是一个后处理组件，用于从不同策略输出的动作中选择适当的系统动作。

C. 影响

Rasa 的系统复杂性给使用 Rasa 的开发者（即应用开发者）和创建 Rasa 的开发者（即系统开发者）带来了挑战。

机器学习供应链带来的复杂性。 Rasa 直接或间接依赖 10 个外部机器学习库。在 GitHub 上使用 TensorFlow 的 355392 个项目中，依赖另外 10 个以上深度学习库的项目不足 100 个（0.03%）[69]。由此可以推断，依赖另外 10 个以上机器学习库的情况也并不常见。对于应用开发者而言，很难了解依赖外部机器学习库的组件的实现细节，更不用说选择合适的组件和参数了。例如，由于 Rasa 中的*Mitie* 特征提取器 缺乏文档，应用开发者需要检查 Mitie 的源代码，才能了解它使用 Dlib API 实现典型相关分析。对于系统开发者而言， [81] 漏洞和依赖关系缺陷[32] 可能会因库版本过时或不兼容而出现。因此，未来的工作应为机器学习赋能系统提供组件及其相应依赖机器学习库的管理支持，类似于传统软件组件分析 [26]。

配置带来的复杂性。 Rasa 包含 29 个组件和数百个参数，配置起来可能极其复杂，容易造成配置错误，进而影响功能和性能。这类配置错误与传统可配置软件系统中发生的情况类似 [74]。此外，为应用开发者的特定对话系统场景寻找最优配置也很困难，这也称为配置债务 [63]。尽管自动机器学习已被广泛用于为特定任务选择合适的机器学习模型和参数，但这些研究都侧重于选择单个机器学习模型，而未考虑多个机器学习模型与规则的组合 [28]。另一项挑战是通过测试大量配置来检测潜在缺陷。现有的机器学习模型测试研究主要侧重于使用预定义的超参数测试单个机器学习模型 [82]。

IV. 研究问题2：交互复杂性分析

A. 方法

图中的工作流 1 仅展示了不同模块之间的大致流程。组件之间交互的细节仍未揭示。我们考虑两个或多个组件之间的交互，其中至少有一个是机器学习组件。交互模式包含一个模块占位符，其
可以通过模块中的组件实例化，从而生成交互实例。例如，模式 (PolicyEnsemble,[Policy]) 可以实例化为 (PolicyEnsemble, TEDPolicy) 或 (PolicyEnsemble, (TEDPolicy, RulePolicy))。为回答 **研究问题2**，我们对组件交互模式以及组件的交互实例进行了定性和定量分析。

步骤 1：提取交互模式。 交互可分为两类：模块间交互和模块内交互。(1) 模块间交互：我们考察了两个相邻模块之间的交互（例如，特征提取器与分词器）。我们分析了组件代码中 Message 的用法

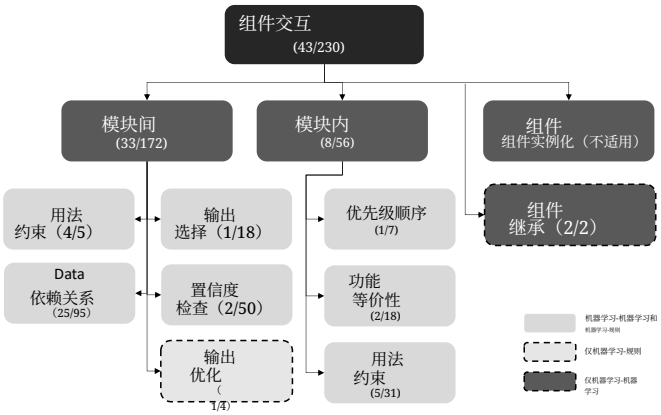


图 2：组件交互分类体系

，因为组件使用 Message 类传输数据。具体而言，我们提取了其上游和下游组件的所有交互模式。我们还将后处理组件（即，回退分类器、实体同义词映射器 和 策略集成）与其所在模块中的其他组件之间的交互视为模块间交互。(2) 模块内交互：我们识别了每个模块中各组件的交互模式。

步骤 2：生成交互实例。对于每种跨模块交互模式，我们将模块占位符分别替换为该模块中的每个组件，以进行实例化。对于每种模块内交互，我们提取每个模块中所有组件的笛卡尔积，作为交互实例。随后，我们筛除不包含机器学习组件或不符合 Rasa 文档所规定约束的组件实例。例如，*CRF* 实体提取器只能使用正则表达式特征提取器的特征，不能使用稀疏特征提取器的特征。

步骤 3：总结交互分类体系。对于生成的组件模式和实例，我们分析了其语义，并总结出组件交互分类体系。

B. 结果

组件交互分类体系如图2所示。它分为 4 个高层类别（即 模块间、模块内、组件实例化和组件继承）以及 8 个子类别。请注意，只有模块间交互包含具有直接数据依赖的组件，而其他类别则包含具有间接交互的组件（例如，两个特征提取器共同使用）。每个类别中的交互模式和交互实例数量列为图模式数量/实例数量（见图 2）。总计有 43 种交互模式和 230 个交互实例。几乎所有类别都包含机器学习组件之间以及机器学习组件与基于规则的组件之间的交互。相比之下，先前关于 Apollo 的研究 [60] 也展示了 8 个子类别中的 4 个，但未提供分类体系和定量分析。

模块间。来自多个模块的组件通过数据传输进行交互。具体而言，输出选择 是指下游组件根据可配置的标准，从多个上游输出中选择合适的输出，例如，结合策略的策略集成 。输出优化 是指

下游组件通过规则完善上游组件的不完美输出，例如，实体抽取器配合实体同义词映射器使用。置信度检查 是指下游组件使用机器学习模型（例如，意图分类器配合 *UnexpecTEDIntent-Policy* 使用）或规则（例如，回退分类器配合意图分类器使用）来检查上游组件输出的可靠性。如果输出被标记为不可靠，则会触发回退行为，例如回退系统动作。使用约束 定义了在特定情况下应当或不应当共同使用的组件。例如，使用 use_lemma 选项时，计数向量特征提取器需要*SpacyTokenizer*；使用 pos_tag 选项时，*LexicalSyntacticFea-turizer* 也需要它。数据依赖包括其余不属于上述任何类别的模块间交互模式，这些交互相对“简单”，没有特定语义。

模块内。模块内组件的交互模式不同于**模块间**。这些组件一起使用时会间接交互。优先级顺序 是指模块内组件的输出按照优先级顺序进行选择，例如策略的优先级顺序。使用约束 与模块间类别中的使用约束 类似。例如，在每个配置文件中，分词器、意图分类器 和实体提取器 中只能使用一个组件，否则其他组件的输出将被覆盖。功能等价 包括不属于上述任何类别的所有模块内交互模式。这些模式是相对“简单”的交互，涉及一起使用但不具备特定语义的组件。

组件实例化。Rasa 支持在配置设置中创建同一组件的多个实例。例如，可以一起使用多个 *CountVectorFeaturizer* 实例（其 n 元语法设置各不相同），以及多个语言模型特征提取器实例（其语言模型各不相同）。我们没有统计此类交互模式和实例，因为开发者可以在配置设置中为一个组件指定无限多个实例。

组件继承。类继承机制使机器学习模型能够在组件之间共享。例如，*UnexpecTED* 意图策略中的机器学习模型定义类是*TED*策略中机器学习模型定义类的子类。

C. 影响

缺乏交互规格说明。 由于机器学习模型具有随机性，因此无法保证机器学习组件针对特定输入会产生何种输出 [8]。因此，与传统系统相比，在机器学习赋能系统中定义交互语义更加困难。当测试样本被错误预测时，要定位确切的故障组件颇具挑战。此外，即使修复了故障组件并提升了其性能，整个系统的总体性能仍可能下降 [78]。因此，训练和评估应从组件级扩展到系统级，以考虑组件之间的交互。总之，我们需要更加重视解决以下问题：

表 II：不同代码类别的代码行数

模块	数据模型规则					
	前置。	后置。	用法定义	用法定义		
分词器	8	80	27	0	25	25
特征提取器	390	323	92	0	162	119
意图分类器	441	131	113	298	3	69
实体提取器	746	311	120	298	24	30
选择器	48	55	9	16	0	0
策略	1332	540	64	543	167	283
共享	996	314	112	1673	0	43
总计	3961	1754	537	2828	381	569

机器学习赋能系统中，因缺乏规格说明而导致的缺陷定位与修复挑战。

隐藏的交互。即使对于 Rasa 的系统开发者来说，识别所有交互也并非易事。例如，数据依赖交互发生在正则表达式特征提取器和 *CR-FEntityExtractor* 之间，但文档中并未标明，只能通过源代码识别。应用开发者可能会误用组件，并因不了解这些隐藏的交互而对系统性能不佳感到困惑，尤其是在使用约束、输出选择和优先级顺序等交互类别中。可以探索数据流分析等技术，以自动揭示机器学习赋能系统中的组件交互 [62]。

此外，我们关于组件交互复杂性的结果有助于指导开发者构建更好的机器学习赋能系统。例如，开发者可以遵循输出选择和输出优化交互模式，在系统层面改进组件的输出；还可以利用置信度检查检测机器学习模型无法处理的情况，进而触发回退规则。这对于自动驾驶系统等安全关键系统尤为重要 [60]。

V. RQ3：组件复杂度分析

A. 方法

为回答**RQ3**，我们对每个组件中的代码片段类别进行分类，并探究其组合模式。

步骤 1：标注代码片段。我们根据语义将每个源代码文件划分为代码片段，然后将其分为 6 类：(1) 模型定义，即机器学习模型的定义代码；(2) 规则定义，即基于规则的组件中规则的定义代码；(3) 模型使用，即机器学习模型的使用代码；(4) 规则使用，即规则的使用代码；(5) 数据预处理，即模型或规则使用前对输入数据进行处理的代码；(6) 数据后处理，即模型或规则使用后对输出数据进行处理的代码。两位作者分别独立标注代码片段，第三位作者参与解决分歧。两位作者的科恩卡帕系数达到 0.830。

步骤 2：总结代码片段的组合模式。基于已标注的代码片段，我们总结了每个组件中数据处理代码以及模型或规则使用代码的组合模式。

B. 结果

不同代码类别的统计数据如表II所示。我们仅统计了带标签代码片段的代码行数，而

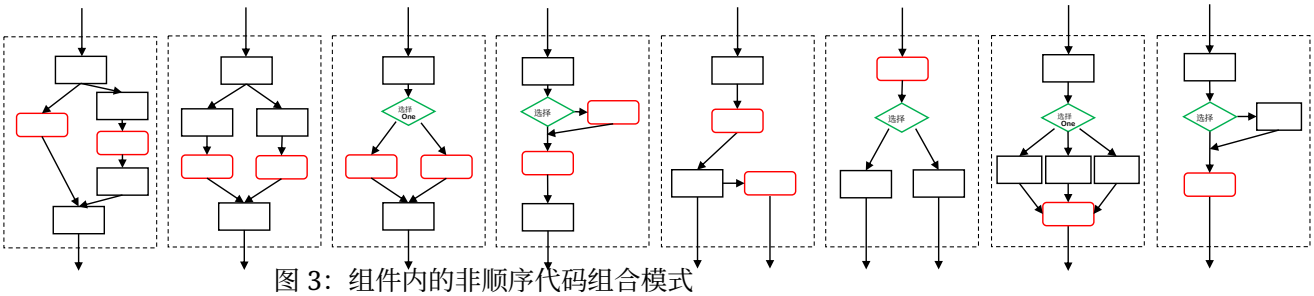


图 3：组件内的非顺序代码组合模式

忽略类初始化等通用工具代码。数据处理代码共计 5715（57.1%）行代码，模型使用与定义代码和规则使用与定义代码分别占 3365（33.5%）行和 950（9.4%）行。在模型定义代码的 2828 行中，有 1673 行（59.2%）位于共享模块中，这表明不同组件之间复用模型定义代码的情况相当普遍。分词器和特征提取器中没有模型定义代码，因为机器学习组件均构建在外部机器学习库之上。

由于 Rasa 中数据处理代码占据主导比例，我们将数据预处理和数据后处理类别进一步划分为更具体的类型。具体而言，验证代码旨在验证组件的输入或输出数据。格式转换 代码会转换数据格式，例如从 Python 数组构造向量并重塑向量。组件输入/输出筛选 代码会筛选不符合指定条件的数据，例如缺少某些属性的数据。数据缩放/填充/编码/解码 代码会改变数据的值，而数据拆分/打乱/平衡/批处理/排序代码会改变数据的组织方式，以便更好地训练和推断组件。我们在网站上提供了完整的代码手册以及数据处理类型的统计信息 [7]。

此外，我们发现代码片段的组合模式包括顺序代码组合模式和各种非顺序组合模式。在典型的顺序组合模式中，数据先经过预处理，再由模型或规则使用代码处理，最后进行后处理。非顺序代码组合模式总结见图 3。黑色方框表示数据处理代码片段，红色方框表示模型或规则使用代码片段，绿色菱形表示选择一个或多个下游代码片段。前 5 种模式由同一组件中的多个模型或规则用法组成。最后 3 种模式则由单个模型或规则用法以及多个可能的数据处理代码片段组成，具体选择取决于配置或输入数据。

C. 影响

数据处理。数据处理代码分散在不同的粒度级别，与机器学习模型和规则中有完善文档且结构清晰的代码不同。具体而言，数据处理代码包括数据处理组件（例如，*PolicyEnsem-ble*）、共享模块中的通用数据处理类和函数，以及与模型或规则用法交织在一起的组件中的特定数据处理代码片段。一方面，应用开发者可能难以识别和理解所有数据处理代码的语义。一个具体的例子是，TransformerRasaModel 的模型定义类中也包含数据预处理代码，包括共振峰转换 和数据批次代码。使用程序分析等技术自动提取并分析数据处理代码的语义可能会很有帮助 [61]。另一方面，对于系统开发者而言，维护和测试数据处理代码可能颇具挑战；随着机器学习开发范式从以模型为中心转向以数据为中心，这可能会导致严重后果 [45]。总的来说，建立数据处理代码的分类体系将有助于维护和测试数据处理代码。

代码组合模式。这些非顺序组合模式可能会给机器学习赋能系统带来额外的动态复杂性，例如，使用静态分析捕获代码片段所有可能的运行时组合，成本过高。尽管动态测试被广泛用于弥补传统软件中静态分析的局限性[24]，，但现有的大多数针对机器学习的测试技术仅针对机器学习模型层面 [82]。将这些技术扩展到数据处理代码和组合模式，将大有裨益。

VI. RQ4：测试实践分析

A. 方法

为回答**RQ4**，我们分三个步骤检查了测试用例。

步骤 1：标注测试用例。我们手动标注了每个测试用例的粒度级别、预言机类型和机器学习阶段。测试用例有三种不同的粒度级别：(1) 方法级：测试单个或多个方法；(2) 组件级：测试组件在训练阶段或推理阶段的完整流程；(3) 集成级：测试当前组件与上游组件的集成。有四种测试预言机类型：(1) 给定的输入输出对：给出输入和预期输出数据；(2) 组件特定约束：必须满足根据组件实现定义的约束，即分类器生成的意图列表中的置信度分数之和应等于 1；(3) 差分执行：不同设置下执行的输出应发生变化或保持不变，即对于相同的输入，原始机器学习模型与其从磁盘加载的版本的输出应保持一致；(4) 异常：测试用例是否会针对特定输入和配置抛出异常。最后，测试用例覆盖的机器学习阶段包括训练阶段、推理阶段和评估阶段。要测试组件的训练阶段，测试用例必须先对其进行训练，然后检查是否违反了任何测试预言。请注意，每个测试用例可能涉及多种预言机类型和机器学习阶段，但只能有一个粒度级别，在

表 III：测试用例的代码覆盖率和标注统计

模块	总计	测试用例类型			测试用例阶段			预言机类型				代码覆盖率	
		方法	组件	集成	推理	训练	评估	I-O	C-S	差异	异常	语句覆盖率	分支覆盖率
分词器	27	7	20	0	25	14	0	24	1	0	3	97.4%	96.8%
特征提取器	62	13	14	35	56	40	0	46	5	3	8	95.7%	94.9%
意图分类器	36	7	15	14	29	30	0	18	11	7	1	92.5%	89.5%
实体提取器	41	6	19	16	34	31	0	18	14	8	1	92.3%	90.1%
选择器	13	4	6	3	9	12	0	6	5	1	1	68.3%	66.4%
策略	165	77	88	0	105	127	0	117	51	0	20	95.7%	94.5%
共享	138	129	2	7	90	84	0	89	42	2	16	92.3%	91.4%
总计	461	240	156	65	331	317	47	312	123	15	49	93.2%	92.0%

表 IV：组件交互的测试覆盖率

类别	子类别	覆盖模式	覆盖实例
模块间	数据依赖	9/25	17/95
	置信度检查	0/2	0/50
	输出选择	0/1	0/18
	输出优化	1/1	1/4
	使用约束	3/3	3/3
模块内	功能等价 2/2		3/18
	优先级顺序 1/1		4/7
	使用约束	2/2	2/4
总计		18/37	30/199

Rasa。两位作者分别独立标注了测试用例，第三位作者参与解决分歧。粒度级别、测试预言和机器学习阶段的科恩卡帕系数分别为 0.907、0.908 和 0.854。

步骤 2：收集测试用例的代码覆盖率。我们使用*pytest-cov*收集了代码的语句覆盖率和分支覆盖率，因为 Rasa 使用 *pytest* 运行测试用例。

步骤 3：收集测试用例的交互模式覆盖率。我们在每个组件的方法中注入日志语句，然后执行测试用例，收集每个测试用例中共同执行的组件集合。我们尝试将所有组件交互实例与这些组件集合进行匹配，但不包括模型继承实例，以及一些无法同时使用的使用约束实例。匹配到的交互模式和实例被视为已由测试用例覆盖。

B. 结果

测试用例的代码覆盖率和标注统计如表 III所示。(1) 代码的语句覆盖率和分支覆盖率总计分别达到 93.2% 和 92.0%，远高于 Apollo 中的 21.5% 和 13.3% [60]。(2) 选择器的覆盖率仅为 68.3% 和 66.4%，因为选择器有两个候选机器学习模型，但只测试了其中一个。(3) 方法级、组件级和集成级测试用例分别有 240 个 (52.0%)、156 个 (33.8%) 和 65 个 (14.1%)。(4) 策略中没有集成级测试用例，因为策略使用开发者提供的给定意图和实体输入进行测试，不依赖自然语言理解模块。(5) 推理阶段和训练阶段的测试用例数量相近。(6) 只有共享模块中的测试用例覆盖了评估阶段，因为共享模块为所有组件提供评估代码。(7) 给定输入输出对、组件特定约束、差分执行和异常测试预言机对应的测试用例分别有 312 个 (67.7%)、123 个 (26.3%)、15 个 (3.3%) 和 49 个 (10.6%)。

如表IV 所示，交互的测试覆盖率相对较低，即 37 种模式中有 18 种（48.6%）以及 199 个实例得到了覆盖。这是因为只有集成级测试会覆盖组件之间的交互。具体而言，置信度检查和输出选择 未得到覆盖。

C. 影响

组件交互的测试覆盖率较低。由于庞大的配置空间和隐藏交互所带来的复杂性，要实现较高的组件交互测试覆盖率十分困难。能够覆盖组件交互的测试用例（即集成级测试用例）所占比例不超过 15%。然而，集成级测试用例能够比组件级和方法级测试用例覆盖并杀死更多变异体，因为许多变异体不会在非集成级测试用例中显现 [42]。因此，为机器学习赋能系统生成集成级测试用例至关重要。

测试预言机类型有限。由于缺乏针对交互的规格说明所带来的复杂性，编写具有给定的输入输出对以及组件特定约束预言机的测试用例既困难又耗时。因此，无需交互规格说明的测试用例，即差分执行和异常测试预言机，已被广泛用于解决传统软件测试用例生成技术中的预言机问题，例如差分测试 [21], 模糊测试 [44] 和基于搜索的测试 [50]。此外，我们发现，使用这两种预言机的测试用例具有与组件特定约束预言机相近的变异体杀伤能力（见 **RQ5**）。尽管如此，Rasa 中只有 13.9% 的测试用例采用了差分执行和异常测试预言机，这表明在机器学习赋能系统的测试用例生成技术中应用这两种测试预言机类型仍有很大空间。

VII. RQ5：变异测试分析

A. 方法

为回答 **RQ5**，我们对变异测试进行了分析 [37]。该方法应用变异算子来生成错误代码的不同版本，即变异体。针对每个变异体，我们执行测试用例并收集测试结果，以判断该变异体是否被杀死。由于 Rasa 同时包含机器学习组件和基于规则的组件，我们既考虑了传统软件的变异算子（即语法变异算子），也考虑了机器学习专用变异算子。如表 V所示，我们使用了 Jia 等人的 9 种语法变异算子 [36] 以及 DeepCrime 的 11 种机器学习专用变异算子 [33]。

我们在下文列出变异分析的步骤。

步骤 1：生成变异体。我们使用 *mutmut* [51]生成了语法变异体。我们使用了两组语法变异算子，即逻辑和值，它们分别对逻辑流程和变量

值进行变异。此外，我们使用 DeepCrime 生成了机器学习特定变异体 [33]。我们使用了 DeepCrime 八个变异组中的四个（激活、正则化、权重和优化）。其他变异组中，训练数据和验证组中的变异算子并非本文关注的重点；未纳入超参数组，因为 Rasa 中的超参数由开发者通过配置文件指定；此外，损失函数组不适用，因为 Rasa 中的损失函数均为从头实现，而 DeepCrime 提供的变异算子仅用于将 Keras 损失函数 API 替换为另一个 API。此外，我们仅针对**RQ3**中标记的 6 类代码生成变异体，不包括通用工具代码。为减少潜在偏差，我们为每个 Python 类生成的变异体不超过 30 个。每个变异体也只修改一个抽象语法树节点。

步骤 2：使用测试用例进行变异测试分析。对于每个变异体，仅收集（来自**RQ4**中的测试覆盖率数据）并执行覆盖了变异行的测试用例，以节省运行时间。如果任何测试用例在某个变异体上失败，则认为该变异体被该测试用例杀死；否则，认为该变异体存活。测试用例可能出现三种失败症状：(1) 断言失败；(2) 出现执行错误或运行时错误；(3) 测试用例超时。测试用例的最长运行时间为其在原始无缺陷代码版本中运行时间的 10 倍。此外，为避免不稳定测试，每个变异体都执行了 3 次测试用例。我们发现，所有测试用例的状态在三次运行中都保持一致。

步骤 3：使用测试数据进行变异测试分析。对于步骤 2 中存活的变异体，我们使用 3 个默认配置文件和 *Multiwoz* [15], 中的餐厅领域数据评估其影响；Multiwoz 是一个广泛用于评估对话系统性能的多领域数据集。给定一个配置文件，只有其中指定的组件会被纳入 Rasa 流水线，因此，步骤 2 中某些存活变异体的变异节点不会执行，因为它们不会受到该配置的影响。由于机器学习程序具有随机性，我们使用随机划分的训练/测试数据（80/20）分别对变异程序和原始程序训练 5 次，并判断两个版本的性能指标是否存在统计显著差异，且效应量不可忽略，也非微小。我们采用与 [33, 35]，相同的公式来判定变异体是否被测试数据杀死，其中显著性水平阈值为 0.05，效应量阈值为 0.5。我们采用意图分类器、实体提取器和策略 的 F1 分数作为性能指标；也就是说，如果三个模块中任意一个模块的 F1 分数在两个代码版本之间存在统计显著差异，则将该变异体标记为被测试数据杀死。

B. 结果

各个变异算子的变异测试结果如表 V 所示。共生成 1447 个变异体，其中 1106 个（76.4%）被测试用例杀死，341 个（23.6%）存活，146 个（10.1%）受到影响，22 个（1.5%）被测试数据杀死。在 341 个存活变异体中，只有 146 个会影响默认的 3 条 Rasa 流水线，这表明庞大的配置空间很难得到充分测试。测试用例杀死了 81.3% 的语法变异体和 20.0% 的机器学习特定变异体，而

表 V：变异测试结果

变异组	运算符	总计	测试用例		测试数据	
			被杀死	存活	受影响	被杀死
逻辑	ArOR	109	86	23	10	1
	ComOR	109	88	21	6	0
	LogOR	145	112	33	14	0
	AsOR	20	19	1	0	0
	MemOR	32	30	2	0	0
	KVR	12	7	5	1	0
值	BVR	64	32	32	9	0
	NVR	224	180	64	18	2
	AsVR	582	525	67	10	0
激活	ACH	22	3	19	18	6
	ARM	2	0	2	1	0
	AAL	22	11	11	11	2
正则化	RAW	6	0	6	3	3
	RCW	10	0	10	10	0
	RRW	5	0	5	5	0
权重	WCI	24	10	14	13	1
	WAB	4	0	4	3	1
	WRB	3	1	2	2	0
优化	OCH	24	2	22	9	6
	OCG	8	0	8	3	0
总计		1447	1106	341	146	22

表 VI：变异体位置结果

位置	总计	测试用例结果		测试数据结果	
		被杀死	存活	受影响	被杀死
数据预处理	385	326	59	23	0
数据后处理	271	222	49	5	0
模型使用	307	243	64	4	0
模型定义	364	224	140	99	22
规则使用	4	4	0	0	0
规则定义	115	101	14	4	0

表 VII：测试用例变异结果

类别	Type	测试数量	强测试数量	覆盖	被杀死
粒度	方法	240	59	947	635
	组件	156	31	1121	709
	集成	65	29	903	613
阶段	推理	331	86	1358	995
	训练	317	75	1184	847
	评估	47	11	772	476
预言机类型	I-O	312	98	1298	956
	C-S	123	19	1103	707
	Diff	15	3	625	338
	异常	49	6	686	352

在受影响变异体中，4.4% 的语法变异体和 24.4% 的机器学习特定变异体被测试数据杀死。这表明，与测试数据相比，测试用例在检测语法变异体方面有效得多，在检测机器学习特定变异体方面则略逊一筹。测试数据杀死的语法变异体和机器学习特定变异体，平均分别导致意图分类器、实体提取器 和 策略的 F1 分数下降 20.8%、0.8%、3.6% 和 11.1%、13.4%、5.7%。

变异体位置相关的变异测试结果如表 VI 所示。模型定义代码中的 364 个变异体中有 224 个（61.5%），其他代码类别中的 1082 个变异体中有 896 个（82.8%）被测试用例杀死。具体而言，除模型定义之外的代码类别中，受测试数据影响并被杀死的变异体很少，这表明测试数据仅对杀死模型定义代码中的变异体有效。

我们通过计算不同测试用例类别中强测试用例数量与测试用例总数的比率，以及其中被杀死的变异体与被覆盖的变异体的比率，研究了检测变异体的能力。我们将强测试

用例定义为能够杀死其覆盖变异体中至少 75% 的测试用例。如表 VII所示，集成级测试用例在三个粒度级别中具有最高的强测试用例比率（44.6%）和最高的被杀死变异体比率（67.9%）。在四种预言机类型中，采用给定输入-输出测试预言的测试用例具有最高的强测试用例比率（31.4%）和最高的被杀死变异体比率（73.7%），而采用其他三种预言机类型的测试用例比率相近。

C. 影响

机器学习赋能系统中的非机器学习特定缺陷和测试用例。数据处理代码带来的复杂性容易引入非机器学习特定缺陷。与测试数据相比，测试用例更能有效地检测语法变异体，即非机器学习特定缺陷。此外，开发者根据测试数据分析、定位和修复机器学习程序中的缺陷并非易事，因此针对机器学习模型的 [85], 调试 [3] 和修复[67] 技术应运而生。开发者通过分析被违反的测试预言，更容易根据失败的测试用例定位和修复缺陷。因此，我们认为，应该更加关注机器学习赋能系统中的非机器学习特定缺陷和测试用例。尽管 Rasa 中有大量测试用例，且代码覆盖率很高，但变异体的杀死率仍有待提升（76.4%），尤其是机器学习特定变异体（29.8%）。现有测试用例生成、选择和质量保证技术在机器学习赋能系统中的适用性与局限性值得进一步探索 [18, 39]。

测试数据在杀死变异体方面的挑战。针对机器学习程序的现有变异测试研究，仅使用测试数据评估变异体，以判断它们能否被杀死 [30, 33, 35,36, 48]。然而，测试数据在大规模机器学习赋能系统中杀死变异体的能力有限，原因有二。首先，由于配置带来的复杂性，只有部分变异体会影响实际配置系统中的组件。其次，训练数据和测试数据的数量及分布会对结果产生很大影响。例如，我们尝试使用原始训练数据的 75% 分别训练无缺陷代码版本和变异版本，被杀死的变异体数量从 22 个变为 83 个，这意味着有些缺陷可能只会在特定训练数据设置下显现。因此，系统开发者应在更多应用开发者可能采用的配置和数据设置下评估并测试机器学习赋能系统，以发现潜在缺陷。

VIII. 威胁

首先，我们对 Rasa 开展了案例研究。Rasa 是一种广泛使用的面向任务的工业对话系统。目前尚不清楚我们的结果能否推广到其他机器学习赋能系统。不过，我们认为，从系统视角研究机器学习赋能系统是一个良好的开端。其次，我们的研究涉及大量对 Rasa 源代码和文档的人工分析，这可能会引入偏差。为减少这些偏差，两位作者分别进行人工分析，第三位作者则参与解决分歧。第三，我们采用的变异算子可能无法模拟现实世界中的缺陷。为缓解这一问题，我们决定使用 DeepCrime 中的变异算子 [33], , 其变异算子实际上是根据现实世界中的机器学习缺陷总结而来的。

IX. 相关工作

机器学习赋能系统研究。尽管许多研究关注机器学习模型，但对系统级分析的关注较少 [38]。Peng 等人 [60] 通过分析机器学习模型如何与系统交互，以及目前开展了哪些测试工作，研究了 Apollo 中机器学习模型的集成。此外，Nahar 等人 [52] 通过访谈探讨了数据科学家与软件工程师之间的协作挑战。Amershi 等人 [5]和 Bernardi 等人 [10] 报告了 Microsoft 和 Booking.com 在机器学习运维方面（从模型需求到模型监控）所面临的挑战与采用的实践。尽管他们仍以模型为中心，但强调模型可能以复杂的方式相互纠缠，导致非单调错误 [5]，而模型质量改进并不一定意味着系统价值提升 [10]。此外，Yokoyama [80]提出了一种将机器学习组件与非机器学习组件分离的架构模式，而 Serban 和 Visser [64]则对机器学习赋能系统的架构挑战进行了综述。Sculley 等人 [63]指出了机器学习赋能系统中的机器学习特有技术债务，Tang 等人 [70]则进一步从真实世界的代码重构中归纳出新的技术债务。此外，也有一些研究尝试解决机器学习组件纠缠问题 [5]，例如，在包含两个机器学习组件的系统上进行蜕变测试 [83], ；依靠人类智慧排查包含三个机器学习组件的系统中的故障 [53],；以及分解包含两个或三个机器学习组件的系统中的错误 [78]。这些研究探讨了模型之间的交互，但仅限于简单系统。此外，Abdessalem 等人 [1, 2] 研究了自动驾驶系统中的特征交互故障，并提出了用于自动检测和修复此类故障的测试与修复方法。Apel 等人 [8] 也讨论了机器学习赋能系统中的特征交互，并提出了应对策略。

与先前工作的主要区别在于，我们以一个大规模复杂机器学习赋能系统为研究对象，从三个层面探索其复杂性，并分析复杂性对测试的影响。最接近的工作是 Peng 等人的 [60], , 但我们报告了更深入的复杂性分析，并开展了测试影响分析。

深度学习模型的变异测试。Jia 等人 [36] 将用于传统程序的语法变异算子应用于深度学习模型。DeepMu- tation [48]和 DeepMutation ++ [30]定义了深度学习特定变异算子。DeepCrime [33] 基于真实的深度学习缺陷，推导出深度学习特定变异算子。Jahangirova 和 Tonella [35] 评估了语法变异算子和深度学习特定变异算子。这些研究聚焦于模型级变异，而我们的目标是系统级变异。

对话系统测试。Bozic 和 Wotawa [13] 提出了一种针对聊天机器人的安全测试方法，以防止跨站脚本攻击和 SQL注入。Bozic 等人 [12] 通过规划测试了一个酒店预订聊天机器人。Bozic 和 Wotawa [14] 为聊天机器人引入了一种蜕变测试方法。类似地，Liu 等人 [47] 使用语义蜕变关系测试对话系统中的自然语言理解模块。尽管已有这些努力，对对话系统进行系统级测试仍未得到足够关注。

X. 结论

我们对 Rasa 进行了全面研究，从三个层面描述其复杂性，并从两个角度探讨复杂性对测试的影响。此外，我们强调了改进机器学习赋能系统软件工程实践的启示。本文使用的所有研究数据和源代码均可在 https://rasasystemcomplexity.github.io/.

参考文献

[1] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand 和 T. Stifter，“使用多目标搜索测试自动驾驶汽车中的特征交互故障”，载于《第33届ACM/IEEE自动化软件工程国际会议论文集》，2018年，第143–154页。[2] ——，“自动驾驶系统中特征交互故障的自动化修复”，载于《第29届ACM SIGSOFT 国际软件测试与分析研讨会论文集》，2020年，第88–100页。[3] A. Abid、M. Yuksekgonul 和 J. Zou，“使用概念性反事实解释，有意义地调试模型错误”，载于《国际机器学习会议论文集》，2022年，第66–88页。[4] A. Aggarwal、P. Lohia、S. Nagar、K. Dey 和 D. Saha，“机器学习模型的黑盒公平性测试”，载于《2019年第27届ACM欧洲软件工程会议与软件工程基础研讨会联合会议论文集》，2019年，第625–635页。[5] S. Amershi、A. Begel、C. Bird、R. DeLine、H. Gall、E. Kamar、N. Nagappan、B. Nushi 和 T. Zimmermann，“机器学习的软件工程：一项案例研究”，载于《第41届国际软件工程会议：软件工程实践论文集》，2019年，第291–300页。[6] Anaconda。（2022年）Dask。 [在线]。网址：https://docs.dask.org/en/ stable/[7] 匿名。（2022年）理解机器学习赋能系统的复杂性及其对测试的影响。 [在线]。网址：https: // rasasystemcomplexity.github.io/ [8] S. Apel、C. K¨astner 和 E. Kang，“强化版特征交互：论机器学习模型的组合”，《IEEE Software》，第39卷，第3期，第120–124页，2022年。[9] T. Baluta、Z. L. Chua、K. S. Meel 和 P. Saxena，“深度神经网络的可扩展定量验证”，载于《第43届国际软件工程会议：配套论文集》，2021年，第248–249页。[10] L. Bernardi、T. Mavridis 和 P. Estevez，“150个成功的机器学习模型：Booking.com的6条经验”，载于《第25届ACM SIGKDD 国际知识发现与数据挖掘会议论文集》，2019年，第1743–1751页。[11] T. Bocklisch、J. Faulkner、N. Pawlowski 和 A. Nichol，“Rasa：开源语言理解与对话管理”，CoRR，第abs/1712.05181卷，2017年。[12] J. Bozic、O. A. Tazl 和 F. Wotawa，“使用人工智能规划测试聊天机器人”，载于《IEEE人工智能测试国际会议论文集》，2019年，第37–44页。[13] J. Bozic 和 F. Wotawa，“聊天机器人的安全测试”，载于《IFIP软件与系统测试国际会议论文集》，2018年，第33–38页。[14] ——，“使用蜕变关系测试聊天机器人”，载于《IFIP软件与系统测试国际会议论文集》，2019年，第41–55页。[15] P. Budzianowski、T.-H. Wen、B.-H. Tseng、I. Casanueva、S. Ultes、O. Ramadan 和 M. Ga˘si´c，“MultiWOZ——用于面向任务的对话建模的大规模多领域绿野仙踪数据集”，载于《自然语言处理经验方法会议论文集》，2018年，第5016–5026页。[16] D. Chaudhuri、A. Kristiadi、J. Lehmann 和 A. Fischer，“通过融入领域知识改进多轮对话系统中的响应选择”，载于《第22届计算自然语言学习会议论文集》，2018年，第497–507页。[17] J. Devlin、M.-W. Chang、K. Lee 和 K. Toutanova，“BERT：用于语言理解的深度双向变换器预训练”，arXiv预印本 arXiv:1810.04805，2018年。[18] D. Di Nardo、N. Alshahwan、L. Briand 和 Y. Labiche，“基于覆盖率的测试用例优先级排序：一项工业案例研究”，载于《2013年IEEE第六届国际软件测试、验证与确认会议，2013年，第302–311页。[19] S. Dola、M. B. Dwyer 和 M. L. Soffa，“利用生成模型对神经网络进行分布感知测试”，载于《第43届 IEEE/ACM 国际软件工程会议论文集》，2021年，第226–237页。[20] S. R. Eddy，“隐马尔可夫模型”，《结构生物学当前观点》，第6卷，第3期，第361–365页，1996年。[21] R. B. Evans 和 A. Savoia，“差分测试：一种变更检测新方法”，载于《第六届欧洲软件工程会议与 ACM SIGSOFT 软件工程基础研讨会联合会议：伴随论文集》，2007年，第549–552页。[22] H. Face。（2022年）Transformer。 [在线]。可访问：https://huggingface. co/docs/transformers/index[23] Facebook。（2022年）Duckling。 [在线]。可访问：https://github.com/facebook/duckling/ [24] R. E. Fairley，“教程：计算机软件的静态分析与动态测试”，《Computer》，第11卷，第4期，第14–23页，1978年。[25] Y. Feng、Q. Shi、X. Gao、J. Wan、C. Fang 和 Z. Chen，“DeepGini：通过对大规模测试进行优先级排序来增强深度神经网络的鲁棒性”，载于《第29届 ACM SIGSOFT 国际软件测试与分析研讨会论文集》，2020年，第177–188页。[26] D. Foo、J. Yeo、H. Xiao 和 A. Sharma，“软件成分分析的动态特性”，CoRR，第abs/1909.00973卷，2019年。[27] Google。（2022年）TensorHub。 [在线]。可访问：https://tensorflow.google. cn/hub[28] X. He、K. Zhao 和 X. Chu，“自动机器学习：最新进展综述”，《Knowledge Based Systems》，第212卷，2021年。[29] M. Henderson、I. Casanueva、N. Mrk˘si´c、P.-H. Su、T.-H. Wen 和 I. Vuli´c，“CONVERT：基于 Transformer 的高效、准确对话表示”，CoRR，2019年。[30] 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《第34届 IEEE/ACM 国际自动化软件工程会议论文集》，2019年，第1158–1161页。[31] Z. Hu、Y. Dong、K. Wang、K.-W. Chang 和 Y. Sun，“GPT-GNN：图神经网络的生成式预训练”，载于《第26届 ACM SIGKDD 国际知识发现与数据挖掘会议论文集》，2020年，第1857–1867页。[32] K. Huang、B. Chen、S. Wu、J. Cao、L. Ma 和 X. Peng，“揭开深度学习技术栈中依赖关系缺陷的面纱”，CoRR，第abs/2207 10347卷，2022年。。[33] 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》，2021年，第67–78页。[34] IBM。（2022年）IBM 2022年全球人工智能采用指数。 [在线]。可访问：https://www.ibm.com/watson/resources/ai-adoption [35] G. Jahangirova 和 P. Tonella，“深度学习系统变异算子的实证评估”，载于《IEEE第13届国际软件测试、验证与确认会议论文集》，2020年，第74–84页。[36] L. Jia、H. Zhong、X. Wang、L. Huang 和 Z. Li，“注入缺陷会如何影响深度学习？”，载于《IEEE国际软件分析、演化与再工程会议论文集》，2022年，第793–804页。[37] Y. Jia 和 M. Harman，“变异测试的发展分析与综述”，《IEEE软件工程汇刊》，第37卷，第5期，第649–678页，2011年。[38] C. K¨astner。（2022年）生产环境中的机器学习：从模型到系统。 [在线]。可访问：https://ckaestne.medium.com/machine-learning-in-production-from-models-to-systems-e1422 7 65 ec[39] cd R. Kazmi、D. N. Jawawi、R. Mohamad 和 I. Ghani，“有效的回归测试用例选择：系统性文献综述”，《ACM 计算调查》（CSUR），第50卷，第2期，第1–32页，2017年。[40] J. Kim、R. Feldt 和 S. Yoo，“使用惊讶度充分性指导深度学习系统测试”，载于《第41届国际软件工程会议论文集》，2019年，第1039–1049页。[41] G. Lample 和 A. Conneau，“跨语言语言模型预训练”，2019 [ ] 1901 07291 。[42] 在线。可访问：https://arxiv.org/abs/。H. Leung 和 L. White，“集成测试与集成级软件回归研究”，载于《软件维护会议论文集 1990》，1990年，第290–301页。

[43] Z. Li、X. Ma、C. Xu、J. Xu、C. Cao 和 J. L¨u，“操作校准：调试现场深度神经网络的置信度错误”，载于《第28届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集》，2020年，第901–913页。[44] H. Liang、X. Pei、X. Jia、W. Shen 和 J. Zhang，“模糊测试：最新进展”，《IEEE可靠性汇刊》，第67卷，1218，2018 3，1199 no. pp. – 。[45] W. Liang、G. A. Tadesse、D. Ho、F.-F. Li、M. Zaharia、C. Zhang 和 J. Zou，“创建可信人工智能数据的进展、挑战与机遇”，《自然·机器智能》2022年。[46] Y. Liu、M. Ott、N. Goyal、J. Du、M. Joshi、D. Chen、O. Levy、M. Lewis、L. Zettlemoyer 和 V. Stoyanov，“RoBERTa：一种鲁棒优化的 BERT 预训练方法”，arXiv 预印本 arXiv:1907.11692，2019年。[47] Z. Liu、Y. Feng 和 Z. Chen，“DialTest：针对循环神经网络驱动的对话系统的自动化测试”，载于《第30届 ACM SIGSOFT 国际软件测试与分析研讨会论文集》，2021年，第115–126页。[48] L. Ma、F. Zhang、J. Sun、M. Xue、B. Li、F. Juefei-Xu、C. Xie、L. Li、Y. Liu、J. Zhao 和 Y. Wang，“DeepMutation：深度学习系统的变异测试”，载于《IEEE第29届国际软件可靠性工程研讨会论文集》，2018年，第100–111页。[49] S. Ma、Y. Liu、W.-C. Lee、X. Zhang 和 A. Grama，“MODE：通过状态差分分析和输入选择实现自动化神经网络模型调试”，载于《2018年第26届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集》，2018年，第175–186页。[50] P. McMinn，“基于搜索的软件测试：过去、现在与未来”，载于《2011年 IEEE第四届软件测试、验证与确认国际会议研讨会》，2011年，第153–163页。[51] Mutmut。（2022年）Mutmut。 [在线]。可访问：https://pypi.org/project/mutmut/[52] N. Nahar、S. Zhou、G. Lewis 和 C. K¨astner，“构建机器学习赋能系统的协作挑战：沟通、文档、工程与流程”，载于《第44届IEEE/ACM国际软件工程会议论文集》，2022年，第413–425页。[53] B. Nushi、E. Kamar、E. Horvitz 和 D. Kossmann，“论人类智慧与机器故障：集成式机器学习系统的故障排查”，载于《第三十一届AAAI人工智能大会论文集》，2017年，第1017–1025页。[54] A. Odena、C. Olsson、D. Andersen 和 I. Goodfellow，“TensorFuzz：利用覆盖率引导的模糊测试调试神经网络”，载于《第36届国际机器学习会议论文集》，2019年，第4901–4911页。[55] K. O’Leary 和 M. Uchida，“利用现有代码创建机器学习流水线时的常见问题”，载于《第三届机器学习与系统会议论文集》，2020年。[56] OpenAI。（2022年）GPT-2。 [在线]。可访问：https://openai.com/blog/tags/ gpt-2/ [57] B. Paulsen、J. Wang 和 C. Wang，“ReluDiff：深度神经网络的差分验证”，载于《第42届ACM/IEEE国际软件工程会议论文集》，2020年，第714–726页。[58] B. Paulsen、J. Wang、J. Wang 和 C. Wang，“NeuroDiff：利用细粒度近似实现可扩展的神经网络差分验证”，载于《第35届 IEEE/ACM 国际自动化软件工程会议论文集》，2020年，第784–796页。[59] K. Pei、Y. Cao、J. Yang 和 S. Jana，“DeepXplore：深度学习系统的自动化白盒测试”，载于《第26届操作系统原理研讨会论文集》，2017年，第1–18页。[60] Z. Peng、J. Yang、T.-H. P. Chen 和 L. Ma，“初探复杂自动驾驶系统中机器学习模型的集成：以 Apollo 为例”，载于《第28届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集》，第1240–1250页。[61] V. Salis、T. Sotiropoulos、P. Louridas、D. Spinellis 和 D. Mitropoulos，“PyCG：Python 中实用的调用图生成”，载于《第43届国际软件工程会议论文集》，2021年，第1646–1657页。[62] F. Sattler、A. von Rhein、T. Berger、N. S. Johansson、M. M. Hardø 和 S. Apel，“将应用间数据流分析扩展至大型应用集合”，《自动化软件工程》，第25卷，第2期，第315–346页，2017年。[63] D. Sculley、G. Holt、D. Golovin、E. Davydov、T. Phillips、D. Ebner、V. Chaudhary、M. Young、J.-F. Crespo 和 D. Dennison，“机器学习系统中隐藏的技术债务”，载于《第28届神经信息处理系统国际会议，2015年，第2503–2511页。[64] A. Serban 和 J. Visser，《使软件架构适应机器学习挑战》，载于《IEEE国际软件分析、演化与再工程会议论文集》，2022年，第152–163页。[65] A. Shmilovici。[66] G. Singh、T. Gehr、M. P¨uschel 和 M. Vechev，《用于认证神经网络的抽象域》，《ACM编程语言》，第3卷，POPL期，第1–30页，2019年。[67] B. Sun、J. Sun、L. H. Pham 和 J. Shi，《基于因果关系的神经网络修复》，载于《第44届国际软件工程会议论文集》，2022年，第338–349页。[68] Y. Sun、M. Wu、W. Ruan、X. Huang、M. Kwiatkowska 和 D. Kroening，《深度神经网络的协同符号测试》，载于《第33届ACM/IEEE自动化软件工程国际会议论文集》，2018年，第109–119页。[69] X. Tan、K. Gao、M. Zhou 和 L. Zhang，《深度学习供应链探索性研究》，载于《IEEE/ACM第44届国际软件工程会议论文集》，2022年，第86–98页。[70] Y. Tang、R. Khatchadourian、M. Bagherzadeh、R. Singh、A. Stewart 和 A. Raja，《机器学习系统中的重构与技术债务实证研究》，载于《IEEE/ACM第43届国际软件工程会议论文集》，2021年，第238–250页。[71] G. Tao、S. Ma、Y. Liu、Q. Xu 和 X. Zhang，《Trader：用于调试循环神经网络的轨迹分歧分析与嵌入调节》，载于《ACM/IEEE第42届国际软件工程会议论文集》，2020年，第986–998页。[72] Y. Tian、K. Pei、S. Jana 和 B. Ray，《DeepTest：自动化测试深度神经网络驱动的自动驾驶汽车》，载于《第40届国际软件工程会议论文集》，2018年，第303–314页。[73] F. Toledo、D. Shriver、S. Elbaum 和 M. B. Dwyer，《用于深度神经网络证伪与验证的分布模型》，载于《第36届IEEE/ACM自动化软件工程国际会议论文集》，2021年，第317–329页。[74] M. Velez、P. Jamshidi、N. Siegmund、S. Apel 和 C. K¨astner，《可配置软件系统性能调试：开发者需求与定制化工具支持》，载于《IEEE/ACM第44届国际软件工程会议论文集》，2022年，第1571–1583页。[75] V. Vlasov、J. E. M. Mosig 和 A. Nichol，《对话变换器》，CoRR，第abs/1910.00486卷，2019年。[76] T.-H. Wen、M. Ga˘si´c、N. Mrk˘si´c、P.-H. Su、D. Vandyke 和 S. Young，《用于口语对话系统的语义条件化长短期记忆网络自然语言生成》，载于《自然语言处理经验方法会议论文集》，2015年，第1711–1721页。[77] J. Williams、A. Raux 和 M. Henderson，《对话状态跟踪挑战赛系列：综述》，Dialogue & Discourse，第7卷，第3期，第4–33页，2016年。[78] R. Wu、C. Guo、A. Y. Hannun 和 L. van der Maaten，《失败的修复：机器学习系统中适得其反的改进》，载于《第35届神经信息处理系统会议论文集》，2021年，第11 745–11 756页。[79] Z. Yang、Z. Dai、Y. Yang、J. Carbonell、R. R. Salakhutdinov 和 Q. V. Le，《XLNet：用于语言理解的通用自回归预训练》，《神经信息处理系统进展》，第32卷，2019年。[80] H. Yokoyama，《提升运行稳定性的机器学习系统架构模式》，载于《IEEE软件架构国际会议伴会论文集》，2019年，第267–274页。[81] A. Zerouali、E. Constantinou、T. Mens、G. Robles 和 J. Gonz´alez-Barahona，《npm软件包依赖中的技术滞后实证分析》，载于《第17届软件复用国际会议论文集》，2018年，第95–110页。[82] J. M. Zhang、M. Harman、L. Ma 和 Y. Liu，《机器学习测试：综述、研究版图与未来方向》，《IEEE软件工程汇刊》，第48卷，第1期，第1–36页，2022年。[83] J. Zhang、X. Jing、W. Zhang、H. Wang 和 Y. Dong，《基于蜕变测试提升ARC系统质量》，载于《系统与软件可靠性国际研讨会论文集》，2016年，第137–141页。[84] P. Zhang、J. Wang、J. Sun、G. Dong、X. Wang、X. Wang、J. S. Dong 和 T. Dai，《通过对抗采样进行白盒公平性测试》，载于《ACM/IEEE第42届国际软件工程会议论文集》，2020年，第949–960页。[85] Y. Zhang、P. Ti˘no、A. Leonardis 和 K. Tang，《神经网络可解释性综述》，《IEEE新兴主题汇刊》

计算智能，第 5 卷，第 5 期，第 726–742 页，2021年。[86] Z. Zhang、R. Takanobu、Q. Zhu、M. Huang 和 X. Zhu，“面向任务的对话系统的最新进展与挑战”，中国科学：技术科学，第 63 卷，第 10 期，第 2011–2027 页，2020年。