报告正文

（一）立项依据与研究内容（4000-8000 字）：

1. 项目的立项依据（研究意义、国内外研究现状及发展动态分析，需结合科学研究发展趋势来论述科学意义；或结合国民经济和社会发展中迫切需要解决的关键科技问题来论述其应用前景。附主要参考文献目录）：

1.1 立项背景

我们正处在一个智能化时代（互联网+及工业 4.0 时代），大数据、云计算和人工智能已经成为这个时代进步的三驾马车，它们分别为智能化时代提供数据、算力和算法层面的支持，从而成为各行各业技术革新和社会发展的重要引擎。世界各国为推动智能化进程，特别是美、英、日以及欧盟等发达国家，制定了相应的发展战略以及行动计划。近年来，为促进大数据、云计算和人工智能的发展，我国密集出台了一系列发展战略、行动计划和支持政策：2015 年，国务院颁布《促进大数据发展行动纲要》，强调数据已成为国家基础性战略资源。李克强总理在两会的政府报告中，提出要“制定互联网+行动计划 ”的要求，推动移动互联网、云计算、大数据、物联网与现代制造业结合，促进电子商务、工业互联网和互联网金融健康发展，引导互联网企业拓展国际市场；2016 年，国家发改委、科技部、工信部、中央网信办联合发布了《“互联网＋”人工智能三年行动实施方案》，提出了三大方向共九大工程，系统地提出了我国在 2016-2018 年间推动人工智能发展的具体思路和内容；2017 年，国务院颁布《新一代人工智能发展规划》，指出要抢抓人工智能发展的重大战略机遇，构筑我国人工智能发展的先发优势，加快建设创新型国家和世界科技强国；同年，工业化信息部发布了《促进新一代人工智能产业发展的三年行动计划（2018-2020 年）》，目的在于深入实施“中国制造 2025”，加快人工智能产业发展，推动人工智能和实体经济深度融合，力争于 2020 年在一系列人工智能标志性产业取得重要突破，在若干重点领域形成国际竞争优势。

在当前全面推进战略性新兴产业及高技术制造业建设的形势下，智能软件系统为提升现代企事业单位生产力水平提供重要支撑，为国民经济的飞速增长和社会的持续稳定发展提供有力保障。智能软件系统是指能够产生人类智能行为

的软件系统，通常通过学习或者自适应等方式获得处理问题的逻辑。在智能化时代背景下，大数据提供的海量数据、云计算带来的超强计算能力以及人工智能算法的不断演进为智能软件系统的飞速发展插上了腾飞的翅膀。目前，智能软件系统在各个领域表现良好甚至达到了人类的水平：**（1）改变人们的日常生活：**谷歌、百度以及亚马逊等公司竞相开发无人驾驶汽车并先后在真实的公路中行驶[1-4]；谷歌公司的 AlphaGo 围棋智能机器人[5,6]依靠深度学习技术战胜了排名世界第一的围棋冠军柯洁及职业九段棋手李世石；IBM 公司的深蓝智能计算系统[6]战胜了国际象棋特级大师加里·卡斯帕罗夫；**（2）提升经济运行水平和效率：**京东的无人仓储依靠智能控制系统实现了 6 倍于人类的思考决策速度及 10 倍于传统人工仓库的货物处理效率[7]；微软亚洲研究院利用智能软件系统来优化现有的航运操作，改善航运业网络运营，提升经济效益[8]；**（3）助力科学研究与发现：**谷歌最新的人工智能 AlphaFold，在一项极其困难的任务中击败了所有的对手，成功根据基因序列预测了生命基本分子-蛋白质的三维结构[9]；医学影像企业 Enlitic 开发了从 X 光照片及 CT 扫描图像中找出恶性肿瘤的图像识别软件，利用深度学习的方法对大量医疗图像数据进行机器学习，自动总结出病症的“特征”以及“模式”[10]。

广泛应用的智能化软件系统为人类带来了极大的便利，然而，其质量仍然存在诸多问题，例如：谷歌的无人驾驶汽车和一辆公共汽车相撞，原因是它希望公共汽车在一系列罕见的条件下停车，然而实际上公共汽车不可能停止[11]；特斯拉的一辆无人驾驶汽车和一辆拖车相撞，原因是拖车的外表颜色和天空相近并且底盘较高[12]；微软旗下的聊天机器人 Tay 变成种族歧视和屠杀支持者[13]。特别是在安全攸关的领域，软件质量缺陷造成的损失往往是难以承受的。因此，在智能化软件系统充溢生活的今天，保障智能软件的质量是一个意义深远的问题。**如何有效保障智能化软件系统正确、高效、可靠地实现其既定任务是一个需要解决的重要问题。**

软件测试是一种广泛采用的软件质量保证手段[14]，通过运行有限的测试用例，比较测试用例的输出与预期输出是否一致来检测软件中潜藏的故障。软件测试包含两项关键的任务，即测试用例的生成和测试结果的判定。测试用例生成为待测软件产生用于检查程缺陷的软件输入及其对应的预期输出，直接影响着软件测试的有效性和效率。代表性的软件测试方面的研究工作包括[15]：

- **基于符号执行的测试用例生成：**使用符号值代替具体值分析程序所有可能的执行路径，并通过约束求解为执行路径生成测试用例[16]。符号执行技术由 King 等人[17]首次提出，其初衷是使用符号值执行程序并以此分析程序所有可能的行为。然而，该技术受制于困难的约束求解过程和庞大的程序分析开

销。为解决约束求解难的问题，研究者开发了多种约束求解器，如 z3[18]、STP[19]、choco[20]等；为解决程序分析开销大的问题，研究者提出了选择符号执行[21]、动态符号执行[22]等。此外，研究者通过符号执行生成高覆盖率的测试用例：Cadar 等人[23]设计并实现了符号执行工具 KLEE，可以为待测程序生成超过 90%代码覆盖率的测试用例，Godefroid 等人[25]在符号执行的约束表达式中引入了符号变量在执行中应满足条件的相关描述，提升了白盒测试中用例生成的有效性与效率。

- **基于模型的测试用例生成：**使用待测程序的形式化模型（有限状态机、UML 模型等）自动为待测程序生成测试用例[25]。基于有限状态机的测试用例生成方法将待测程序所有可能的状态进行抽象，定义状态转换所涉及的输入及输出，构造待测程序的有限状态机模型，最后以状态覆盖、转移覆盖、路径覆盖等策略从状机模型中选择状态转移序列并生成测试用例[15][26]。例如，Kansomkeat 和 Rivepiboon [27]使用 UML 状态图为待测程序构建模型并自动生成满足一定覆盖条件的测试用例集。
- **基于组合测试的测试用例生成：**使用不同的组合策略将待测程序不同参数的值进行组合，为待测程序生成规模尽可能小而不显著损失故障检测能力的测试用例集（组合覆盖表）。目的是使用较少的测试用例分析软件中各参数之间的相互作用对整个系统产生的影响。目前，组合测试中主要有四类测试用例生成方法，包括贪心算法、启发式搜索算法、代数方法、随机方法。贪心算法以尽可能多地覆盖未覆盖组合为原则给待测程序逐一生成测试用例并形成测试集，典型的方法有 one-row-at-a-time [28]及 in parameter order 算法[29,30]。启发式规则对预先存在的测试集进行一系列转换，直至测试集覆盖所有的组合，典型的方法有 Ghazi 等人[31]提出的基于遗传算法的技术及 Bryce 和 Colbourn[32]在贪心算法基础上改进的技术。代数方法通过数学函数或递归构造的方式推导测试用例，典型的方法有[33,34]。随机方法则以随机的方式从测试全集中选择测试用例，通常被作为实验对比的基准方法[35-37]。
- **适应性随机测试：**使新生成的测试用例与所有已生成测试用例具有最远的距离，使测试用例在软件输入域上更加均匀地分布。Chen 等人[38]在此想法的基础上提出了适应性随机测试，改进了随机测试的故障检测能力。此后，研究者在最远距离测试用例选取方面提出了多种不同的策略，形成了适应性随机测试的多个变种。Chen 等人[38]从已有的候选随机测试用例中选取与已生成测试用例差别最大的测试用例，并提出了基于固定规模候选集的适应性随机测试（FSCS-ART）。Chen 等人[39]提出了镜像适应性随机测试（MART），

通过在输入域中选择与已有测试用例“镜像”的新测试用例，实现最远距离测试用例的选取，，减少了适应性随机测试的计算开销。Chen 等人[40]提出了基于随机分区的适应性随机测试，使用已执行测试用例对程序的输入域进行划分，并从最大的分区中选取测试用例。。

- **基于搜索的测试用例生成：**使用搜索算法自动化从待测程序的输入空间中寻找能够最大程度满足测试目标且最大限度降低测试成本的测试用例集。在程序结构测试方面，研究者针对多种程序覆盖准则提出了不同的测试用例生成技术[51]，如分支覆盖[52-53]、判定覆盖[54]、数据流覆盖[55-56]等。在基于模型的测试方面，程序模型的状态迁移覆盖一直是研究的热点，代表性的工作有：Li 等人[57-58]为 UML 状态图表达的程序模型生成满足状态迁移覆盖的测试用例集，Lefticaru 和 Ipate[59]使用遗传算法自动为程序的状态图生成测试用例集。在变异测试方面，研究者通过搜索的手段生成能够杀死待测程序变异体的测试用例[60-63].
- **动态随机测试技术：**Cai 等人将控制论引入到软件测试中，提出了利用测试过程的历史信息更新测试剖面的适应性测试技术[101]，但是适应性测试技术的时间开销太高。为了降低适应性测试技术的时间开销，Cai 等人根据引起故障的输入趋向于集簇在连续的区域这一观察[102,103]，提出了利用当前的测试信息更新测试剖面的动态随机测试技术[104]。该技术的主要特点是在测试的过程中根据每一次的执行结果动态改变测试剖面，使得具有较高失效率的分区的选取概率变大。动态随机测试的性能受两方面的影响：初始测试剖面和动态随机测试中的参数值。这方面代表性工作包括：（a）初始测试剖面：Li 等人提出了在测试过程中出现预定标准时测试剖面转换成理论最优剖面的最优动态随机测试技术[105]。（b）参数值：Yang 等人提出了在测试的过程中动态调整参数的方法[106]；Lv 等人通过理论分析的方式分析两个参数的比值在一定的区间时可以保证失效率大的分区选取概率不断增大[107]

测试结果判定是检查测试用例对应的实际输出是否符合预期的步骤，其结果是判断待测软件中是否潜藏故障的重要依据。然而，在很多实际情况中，难以验证测试用例对应的实际输出结果是否满足预期，此问题称为软件测试中的“测试预期问题”。测试预期问题一直是软件测试领域中的热点与难点问题[106]，代表性的研究工作包括：N-version 测试[64]、断言[65]、机器学习[66]、假设检验[67]、蜕变测试[68]。其中，蜕变测试依据待测软件的蜕变属性获取蜕变关系，执行原始测试用例（采用测试用例生成技术获得）与衍生测试用例（根据蜕变关系获得），检查对应的输出是否违反蜕变关系。如果违反了某种蜕变关系，则表明存在故障。不难看出，蜕变测试无需测试预期，因此有效地缓解了测试预

期问题，并在多个领域得到成功的应用[69]。近年来，研究者在蜕变测试理论与应用等方面取得了丰富的研究成果。部分代表性工作如下：

- **蜕变关系识别与复合方面：**Chen 等人提出了一种基于范畴划分的蜕变关系识别方法[70]；Zhang 等提出一种基于搜缩的蜕变关系推理方法[71]；Su 等人提出了一种蜕变关系动态识别方法[72]；Sun 等人提出一种数据变异指导的蜕变关系获取方法[73]；Gotlieb 等人提出了一种蜕变关系验证框架[74]，对于给定的程序的某个蜕变关系植入故障，利用约束逻辑编程技术生成满足植错后的蜕变关系的测试用例集，如果测试用例集检测出故障，则表明植错后的蜕变关系是错误的；Liu 等人提出一种蜕变关系复合方法，通过对多个蜕变关系的复合形成新的蜕变关系[75]。
- **蜕变测试原始测试用例生成方面：**Batra 等人提出了一种基于遗传算法的测试用例生成方法，旨在最大化覆盖程序路径[76]；Dong 等人采用符号执行提取蜕变关系并生成相应的测试用例[77]；Chen 等人比较了特殊值法与随机方法产生的原始测试用例对蜕变测试效率的影响[78]。
- **蜕变测试与其它测试技术结合：**Xie 等人将蜕变测试与基于频谱分析的故障定位技术结合，提出了无需预期的故障定位方法[79]； Dong 等人将蜕变测试与遗传算法相结合，在搜索过程中使用蜕变关系，将原始测试用例和衍生测试用例都视为候选方案，加速达到某种覆盖标准[80]。
- **蜕变测试在不同领域中的应用：** Sun 等人提出了面向 Web 服务的蜕变测试方法 [81-82]，首先从 Web 服务的 WSDL 描述文档中提取蜕变关系，然后根据 WSDL 文档随机产生测试用例并且运用蜕变关系得到相应的衍生测试用例；Chan 等人提出了一种面向 SOA 的蜕变测试方法，将离线测试通过的测试用例作为原始测试用例，在线测试用例作为衍生测试用例[83-84]；Mayer 等人将蜕变测试应用于图像处理程序[85]；Kuo 等人将蜕变测试应用于图象处理程序时检测出一个真实故障[86]；Tse 等人尝试将蜕变测试应用于上下文敏感的中间件软件[87]；Chan 等人将蜕变测试应用于能量感知的无线传感器网络应用软件[88]；Sun 等人将蜕变测试成功地用于几类典型加密程序的测试[89]。

与传统软件相比，智能软件系统呈现出**学习深层知识、融合跨界数据、数据处理层级化、决策逻辑不受控、系统输出难以验证、集成群体智慧**的特点，具体来说：

- **在学习深层知识方面：**智能化软件系统能够从大数据表示的知识中进行深层次的认知、学习、推理，并通过动态调整自身业务逻辑实现持续演化，适应外部动态变化的需求，而传统软件依据既定的业务逻辑执行各种操作，难以

进行自主演化并适应外部环境变化。

- **在融合跨界数据方面：**智能化软件系统将跨领域的多源异构数据进行协同处理，实现跨界信息的关联融合，而传统软件系统仅仅分类型处理这些多源异构数据，难以实现信息的深度融合；多领域的数据进行组合导致智能软件系统的输入空间异常庞大，此外，大量异构的数据造成智能软件系统数据预处理模块异常复杂。
- **在数据处理方面：**智能软件系统首先采集数据，并对采集的数据进行预处理，得到符合决策模块输入类型的规则数据，然后将规则数据输入决策模块，并得到智能软件系统的输出。智能软件系统多层次的数据处理过程导致数据出错的可能性增加。
- **在获得决策逻辑方面：**智能软件系统的决策模块通常由程序开发人员利用高级语言（Java、Python 等）实现，但是决策的逻辑从数据中习得，而不是开发人员指定。这种“在代码上堆代码”的方式让系统开发人员以及测试人员难以理解。
- **在验证测试结果方面：**智能软件系统多源化的输入、难以理解的内部逻辑以及所处场景的多样性使得测试人员判断系统行为耗时、耗力。
- **在集成群体智慧方面：**智能化软件系统可依托互联网或大数据无缝整合多种智能，形成群体智慧，而传统软件系统更加聚焦于个体的智能。

由于上述新特点，**智能软件系统的测试面临诸多新的问题与挑战，具体来说：**

- **测试充分性无法保证：**智能软件系统的输入空间十分巨大，例如无人驾驶汽车通过各种传感器获取无数种可能的外界信息。目前的测试技术主要通过三种方式获得测试数据：**（a）随机生成测试用例：**随机地在测试用例集中挑选测试用例或者随机地生成测试数据具有简单、易用的特点。该方法被谷歌等世界著名公司使用来生成无人驾驶车的测试数据[90-92]。然而，随机的方式没有利用任何系统内部信息以及测试过程信息，使得很多智能系统不正确的行为不能被发现[93-94]；**（b）人工生成测试用例：**该方式通过在普通测试数据的基础上做微小改动来得到新的测试数据[95-96]，然后测试人员判断每一个新测试数据对应的行为。这种方式成功地运用在多种智能软件系统中[97-100]。随机生成和人工生成测试用例的方式都没有考虑智能软件系统的内部结构，不能覆盖智能软件系统的大部分逻辑，导致了这些测试用例只能发现少量的系统异常行为。Pei 等人通过经验研究发现：随机生成一个测试用例几乎能够覆盖无人驾驶车的所有代码，但是只激活了不到 10%的神经元。基于上述观察，Pei 等人提出了神经元覆盖指标，评估测试用例集激活的神经元数目[94]。Tian 等人利用神经元覆盖指标，生成测试用例，激活智

能系统大多数的神经元，并通过经验研究的方式验证该方法生成的测试用例集可以发现更多智能系统不正确的行为；**（c）基于场景和功能：**基于场景的测试技术通过给定的场景及任务测试智能软件系统的实际表现。例如，前文提到的 DARPA 无人驾驶挑战赛中，无人驾驶汽车需要在指定时间内安全地穿越莫哈韦沙漠中的一个区域。对于一些较为简单的智能软件系统的测试，测试人员可以枚举测试场景及任务以验证系统在这些场景下的表现是否符合预期。但是，对于复杂的智能软件系统，穷举测试场景与任务是十分耗时耗力且效率低下的。并且，在组合场景与任务下出现的组合爆炸问题使这种技术难以应用。此外，这种技术的测试结果仅从宏观层面对智能软件系统进行定性评价[44]，难以从微观层面对智能软件系统的功能质量进行定量评价。基于功能的测试技术将智能软件系统按功能划分子模块，并针对每个子模块生成测试用例并针对功能进行测试。例如，一个无人驾驶系统可以被划分为感知及识别模块、决策模块及动作执行模块[45][46]。该技术为这些模块分别生成测试用例，并对模块的功能正确性进行检验。

- **测试用例不可靠：**智能软件系统的测试数据可能不准确并且基于学习的智能软件系统的学习模型经常出现过拟合的现象，出现不正确或者不被期待的行为。
- **数据预处理和决策模块的实现不可靠：**多源、复杂的数据使得预处理模块规模庞大，并且异常复杂的学习模型为开发人员正确实现决策模块增加了难度。
- **难以判定系统行为是否正确：**目前主要的判定有两种：（a）将系统的行为与执行数据对应的标签对比；（b）测试人员判断。这两种方式都需要人工参与，然而在长时间的工作下，资深测试工程师的准确性也不能得到保证。
- **智能软件系统的测试任务难以准确描述：**传统软件系统的运行时功能依据明确定义的规格说明实现，因此，软件测试人员可以根据规格说明清晰且准确地了解与描述系统的待测功能，并以此制定测试计划与任务。然而，智能软件系统具有认知、学习、推理的能力，其功能随着应用场景与任务的变化、认知范围的增加和学习内容的改变而发生变化与衍进，难以预测且呈现出智能性。这些问题导致软件测试人员难以清晰而准确地了解与描述其待测功能。在此情况下，软件测试人员难以对智能软件系统制定测试任务，从而难以实施有组织有目标的软件测试。
- **智能软件系统的模拟测试平台难以搭建：**智能化软件系统通常与外界环境存在密切的交互，所以智能软件系统的测试无可避免地涉及到与环境的交互。然而，在真实环境中测试智能软件系统将产生高昂且难以负担的成本。一个

较好的解决方案是搭建智能软件系统的模拟环境测试平台。如何让模拟测试平台接近智能软件系统的真实应用场景是一个重要的问题。通常，模拟测试平台需要模拟三大类事物，包括人、人造事物、自然事物。 模拟人与自然事物存在诸多困难，其原因是人类在不同场景下的行为是不确定的，很多自然事物的规律及内在机制至今仍未研究透彻。因此，搭建智能软件系统的模拟测试平台存在诸多困难。

- **智能软件系统的质量评价指标及模型存在缺失：**对于传统的软件系统，测试人员可根据 McCall 质量模型、ISO/IEC 25010 质量模型等成熟的软件质量模型为待测系统选定质量特性，并评估系统在测试中（的表现是否）满足给定需求或特性的程度。智能软件系统与传统软件系统最大的不同是前者所特有的人工智能，因此，对智能软件系统的质量进行评价不可不免地要涉及到对系统智能程度的评估。现有的智能度评估方法主要有三类[109]，包括人工辨别法、基准任务法、伙伴对抗法，然而这三类方法分别存在实验设置困难、评估结果片面、结果依赖于对矿伙伴的问题，且未有较为系统的评价指标及模型。有研究者提出通过评价智能系统与人类在行为上的相似性来评估智能系统的质量[48][49]，但仍然没有合适的度量标准。此外，很多智能软件系统是多目标的，例如，无人驾驶系统会考虑驾乘舒适性、燃油消耗等。智能软件系统在多目标情景下会依据目标优先级的不同而展现出不同的行为[42]。在缺乏针对性质量评价指标及模型的情况下，难以对智能软件系统的质量进行系统有效的评估。

综上所述，智能软件系统测试方面的研究工作仍存在诸多不足，**亟待探索有效且高效的新型智能软件系统的测试方法与技术。本课题旨在针对智能软件系统测试的测试方面的问题与挑战，探索智能软件系统的测试关键技术，以系统有效的方式检测与评估智能软件系统的质量，为开发可靠的智能软件系统提供新型测试理论与工具支持。**

参考文献

[1] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, J. K. Hedrick. Learning a Deep Neural Net Policy for End-to-End Control of Autonomous Vehicles. *Proceedings of the 2017 American Control Conference* (ACC'17), IEEE Computer Society, 2017, pp. 4914-4919.

[2] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition* (CVPR'16), IEEE Computer Society, 2016, pp. 770-778.

[3] M. A. O. Vasilescu, D. Terzopoulos. Multilinear Image Analysis for Facial Recognition. *Proceedings of the 16th In International Conference on Pattern Recognition* (ICPR'02), IEEE Computer Society, 2002, pp. 511-514.

[4] W. Xiong, J. Droppo, X. Huang, F. Seide, M. L. Seltzer, A. Stolcke, G. Zweig. Toward Human Parity in Conversational Speech Recognition. *Proceedings of the IEEE/ACM Transactions on Audio, Speech and Language Processing* (TASLP'17), IEEE Computer Society, 2017, pp. 2410-2423.

[5] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489.

[6] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge[J]. Nature, 2017, 550(7676): 354-359.

[7] https://www.iyiou.com/p/73145.html.

[8] http://m.elecfans.com/article/674086.html.

[9] https://deepmind.com/blog/alphafold/.

[10] https://search.proquest.com/docview/2015377429?pq-origsite=gscholar.

[11] http://www.theverge.com/2016/2/29/11134344/google-selfdriving-car-crash-reprt A Google Self-Driving Car Caused a Crash for The Frst Time.

[12] https://electrek.co/2016/07/01/understandingfatal-tesla-accident-autopilot-nhtsa-probe/, Understanding the Fatal Tesla Accident on Autopilot and the NHTSA Probe.

[13] https://en.wikipedia.org/wiki/Tay_(bot).

[14] G. J. Myers, C. Sandler, T. Badgett. The Art of Software Testing. John Wiley and Sons, 2011.

[15] Anand S, Burke E K, Chen T Y, et al. An orchestrated survey of methodologies for automated software test case generation[J]. *Journal of Systems and Software*, 2013, 86(8): 1978-2001.

[16] Baldoni R, Coppa E, D'elia D C, et al. A survey of symbolic execution techniques[J]. *ACM Computing Surveys* (CSUR), 2018, 51(3): 50:1-50:39.

[17] King J C. A new approach to program testing[C]. In *Proceedings of the International Conference on Reliable Software*. ACM, 1975, 228-233.

[18] De Moura L, Bjørner N. Z3: An efficient SMT solver[C]. In *Proceedings of the 14th International conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'08). Springer, 2008: 337-340.

[19] Ganesh V, Dill D L. A decision procedure for bit-vectors and arrays[C]// In

Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07). Springer, 2007: 519-531.

[20] Choco Solver [EB/OL]. [2018-11-18]. http://www.choco-solver.org/.

[21] Chipounov V, Georgescu V, Zamfir C, et al. Selective symbolic execution[C]// In *Proceedings of the 5th Workshop on Hot Topics in System Dependability* (HotDep). 2009.

[22] Odefroid P, Klarlund N, Sen K. DART: directed automated random testing[C]. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI'15). ACM, 2005, 213-223.

[23] Cadar C, Dunbar D, Engler D R. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs[C]// In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (OSDI'08). 2008, 209-224.

[24] Godefroid P, Levin M Y, Molnar D A. Automated whitebox fuzz testing[C]// In *Proceedings of the 15th Annual Network and Distributed System Security Symposium* (NDSS'08). 2008, 151-166.

[25] Dias Neto A C, Subramanyan R, Vieira M, et al. A survey on model-based testing approaches: a systematic review[C]. In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, in conjunction with the *22nd IEEE/ACM International Conference on Automated Software Engineering* (ASE'07). ACM, 2007: 31-36.

[26] Lee D, Yannakakis M. Principles and methods of testing finite state machines-a survey[C]. In *Proceedings of the IEEE*. IEEE Computer Society, 1996, 84(8): 1090-1123.

[27] Kansomkeat S, Rivepiboon W. Automated-generating test case using UML statechart diagrams[C]// In *Proceedings of the 2003 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology* (SAICSIT'03). South African Institute for Computer Scientists and Information Technologists, 2003: 296-300.

[28] Cohen D M, Dalal S R, Fredman M L, et al. The AETG system: An approach to testing based on combinatorial design[J]. *IEEE Transactions on Software Engineering*, 1997, 23(7): 437-444.

[29] Lei Y, Kacker R, Kuhn D R, et al. IPOG: A general strategy for t-way software testing[C]//In *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems* (ECBS'07). IEEE Computer Society, 2007: 549-556.

[30] Lei Y, Kacker R, Kuhn D R, et al. IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing[J]. *Software Testing, Verification and Reliability*, 2008, 18(3): 125-148.

[31] Ghazi S A, Ahmed M A. Pair-wise test coverage using genetic algorithms[C]. In *Proceedings of the Congress on Evolutionary Computation* (CEC'03). IEEE Computer Society, 2003, 1420-1424.

[32] Bryce R C, Colbourn C J. One-test-at-a-time heuristic search for interaction test suites[C]. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (GECCO'07). ACM, 2007: 1082-1089.

[33] Williams A W. Determination of test configurations for pair-wise interaction coverage[M]. Testing of Communicating Systems. Springer, Boston, MA, 2000: 59-74.

[34] Kobayashi N, Tsuchiya T, Kikuno T. A new method for constructing pair-wise covering designs for software testing[J]. *Information Processing Letters*, 2002, 81(2): 85-91.

[35] Calvagna A, Fornaia A, Tramontana E. Random versus combinatorial effectiveness in software conformance testing: A case study[C]. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing* (SAC '15). ACM, 2015: 1797-1802.

[36] Vilkomir S, Starov O, Bhambroo R. Evaluation of t-wise approach for testing logical expressions in software[C]. In *Proceedings of the 6th International Conference on Software Testing, Verification and Validation Workshops* (ICSTW'13). IEEE, 2013: 249-256.

[37] Medeiros F, Kästner C, Ribeiro M, et al. A comparison of 10 sampling algorithms for configurable systems[C]. In *Proceedings of the 38th International Conference on Software Engineering* (ICSE'16). ACM, 2016: 643-654.

[38] Chen T Y, Leung H, Mak I K. Adaptive random testing[C]. In *Proceedings of the 9th Annual Asian Computing Science Conference* (ASIAN'04). Springer, 2004: 320-329.

[39] Chen T Y, Kuo F C, Merkel R G, et al. Mirror adaptive random testing[J]. *Information and Software Technology*, 2004, 46(15): 1001-1010.

[40] Chen T Y, Merkel R, Wong P K, et al. Adaptive random testing through dynamic partitioning[C]. In *Proceedings of the 4th International Conference on Quality Software* (QSIC'04). IEEE, 2004: 79-86.

[41] Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. *IEEE Transactions on Software Engineering*, 2015, 41(5): 507-525.

[42] Li L, Lin Y L, Zheng N N, et al. Artificial intelligence test: a case study of intelligent vehicles[J]. *Artificial Intelligence Review*, 2018: 1-25.

[43] Campbell M, Egerstedt M, How J P, et al. Autonomous driving in urban environments: approaches, lessons and challenges[J]. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 2010, 368(1928): 4649-4672.

[44] Li L, Huang W L, Liu Y, et al. Intelligence testing for autonomous vehicles: a new approach[J]. *IEEE Transactions on Intelligent Vehicles*, 2016, 1(2): 158-166.

[45] Huang W L, Wen D, Geng J, et al. Task-specific performance evaluation of UGVs: case studies at the IVFC[J]. *IEEE transactions on Intelligent Transportation Systems*, 2014, 15(5): 1969-1979.

[46] Li L, Wen D, Zheng N N, et al. Cognitive cars: A new frontier for ADAS research[J]. *IEEE Transactions on Intelligent Transportation Systems*, 2012, 13(1): 395-407.

[47] Pei K, Cao Y, Yang J, et al. DeepXplore: Automated whitebox testing of deep learning systems[C]//In *Proceedings of the 26th Symposium on Operating Systems Principles* (SOSP'17). ACM, 2017: 1-18.

[48] Argall B D, Chernova S, Veloso M, et al. A survey of robot learning from demonstration[J]. *Robotics and autonomous systems*, 2009, 57(5): 469-483.

[49] Kuefler A, Morton J, Wheeler T, et al. Imitating driver behavior with generative adversarial networks[C]// In *Proceedings of IEEE Intelligent Vehicles Symposium* (IV). IEEE, 2017: 204-211.

[50] Tian Y, Pei K, Jana S, et al. DeepTest: Automated testing of deep-neural-network-driven autonomous cars[C]. In *Proceedings of the 40th International Conference on Software Engineering* (ICSE'18). ACM, 2018:

303-314.

[51] Harman M, Mansouri S A, Zhang Y. Search based software engineering: A comprehensive analysis and review of trends techniques and applications[R]. Technical Report TR-09-03, Department of Computer Science, King's College London, 2009.

[52] Ahmed M A, Hermadi I. GA-based multiple paths test data generator[J]. *Computers & Operations Research*, 2008, 35(10): 3107-3124.

[53] Alshraideh M, Bottaci L. Search-based software test data generation for string data using program-specific search operators[J]. S*oftware Testing, Verification and Reliability*, 2006, 16(3): 175-203.

[54] Xiao M, El-Attar M, Reformat M, et al. Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques[J]. *Empirical Software Engineering*, 2007, 12(2): 183-239.

[55] Girgis M R. Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm[J]. *Journal of Universal Computer Science*, 2005, 11(6): 898-915.

[56] Ghiduk A S, Harrold M J, Girgis M R. Using genetic algorithms to aid test-data generation for data-flow coverage[C]//In *Proceedings of the 14th Asia-Pacific Software Engineering Conference* (APSEC'07). IEEE, 2007: 41-48.

[57] Li H, Lam C P. An ant colony optimization approach to test sequence generation for state-based software testing[C]// In *Proceedings of the 5th International Conference on Quality Software* (QSIC'05). IEEE Computer Society, 2005, 255-264.

[58] Li H, Lam C P. Using anti-ant-like agents to generate test threads from the UML diagrams[C]//In *Proceedings the 17th IFIP International Conference on Testing of Communicating Systems* (TestCom'05). Springer, 2005: 69-80.

[59] Lefticaru R, Ipate F. Automatic state-based test generation using genetic algorithms[C]//In *Proceedings of the 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (SYNASC'07). IEEE, 2007: 188-195.

[60] Emer M C F P, Vergilio S R. GPTesT: A testing tool based on genetic programming[C]//In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (GECCO'02). Morgan Kaufmann Publishers Inc., 2002: 1343-1350.

[61] Masud M, Nayak A, Zaman M, et al. Strategy for mutation testing using genetic algorithms[C]//In *Proceedings of Canadian Conference on. Electrical and Computer Engineering*. IEEE, 2005: 1049-1052.

[62] Baudry B, Fleurey F, Jézéquel J M, et al. Automatic test case optimization: A bacteriologic algorithm[J]. *IEEE Software*, 2005, 22(2): 76-82.

[63] Baudry B, Fleurey F, Jézéquel J M, et al. From genetic to bacteriological algorithms for mutation-based testing[J]. *Software Testing, Verification and Reliability*, 2005, 15(2): 73-96.

[64] S. S. Brilliant, J. C. Knight, P. E. Ammann. On the Performance of Software Testing Using Multiple Versions. *Proceedings of the 20th international symposium on fault-tolerant computing* (FTCS'90), IEEE Computer Society, 1990, pp. 408–415.

[65] K. Y. Sim, C. S. Low, F. C. Kuo. Eliminating Human Visual Judgment from Testing of Financial Charting Software. *Journal of Software*, 2014, 9(2): 298-312.

[66] W. K. Chan, S. C. Cheung. PAT: A Pattern Classification Approach to Automatic Reference Oracles for the Testing of Mesh Simplification Programs. *Journal of Systems and Software*, 2009, 82(3), 422–434.

[67] R. Guderlei, J. Mayer. Statistical Metamorphic Testing Testing Programs with Random Output by Means of Statistical Hypothesis Tests and Metamorphic Testing. *Proceedings of the 7th international conference on quality software* (QSIC'07), IEEE Computer Society, 2007, pp. 404–409.

[68] T. Y. Chen, S. C. Cheung, S. M. Yiu. Metamorphic Testing: A New Approach for Generating Next Test Cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

[69] S. Segura, G. Fraser, A. B. Sanchez, A. R. Cortes. A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, 2016, 42(9): 805-824.

[70] T. Y. Chen, P. L. Poon, X. Xie. METRIC: Metamorphic Relation Identification Based on the Category-Choice Framework. *Journal of Systems and Software*, 2016, 116: 177-190.

[71] J. Zhang, J. Chen, D. Hao. Search-Based Inference of Polynomial Metamorphic Relations. *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (ASE'14), ACM Press, 2014, pp. 701-712.

[72] F. H. Su, J. Bell, C. Murphy. Dynamic Inference of Likely Metamorphic Properties to Support Differential Testing. *Proceedings of the 10th International Workshop on Automation of Software Test* (AST'15), Co-located with the *37th IEEE International Conference on Software Engineering* (ICSE'15), IEEE Computer Society, 2015, pp. 55-59.

[73] C. Sun, Y. Liu, Z. Wang, W.K. Chan. μMT: A Data Mutation Directed Metamorphic Relation Acquisition Methodology. *Proceeding of the First International Workshop on Metamorphic Testing* (MET 2016), collocated with ICSE 2016, IEEE Computer Society, 2016, pp.12-18.

[74] A. Gotlieb, B. Botella. Automated Metamorphic Testing. *Proceedings of the 27th Annual International Conference on Computer Software and Applications* (COMPSAC'03), IEEE Computer Society, 2003, pp. 34-40.

[75] H. Liu, X. Liu, T. Y. Chen. A New Method for Constructing Metamorphic Relations. *Proceedings of the 12th International Conference on Quality Software* (QSIC'12), IEEE Computer Society, 2013, pp. 59-68.

[76] G. Batra, J. Sengupta. An Efficient Metamorphic Testing Technique Using Genetic Algorithm. *Proceedings of 5th International Conference on Information Intelligence, Systems, Technology and Management* (ICISTM'11), Springer, 2011, pp. 180-188.

[77] G. Dong, T. Guo, P. Zhang. Security Assurance with Program Path Analysis And Metamorphic Testing. *Proceedings of the 4th International Conference on Software Engineering and Service Science* (ICSESS'13), IEEE Computer Society, 2013, pp. 193-197.

[78] T. Y. Chen, F. C. Kuo, Y. Liu. Metamorphic Testing and Testing with Special Values. *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (SNPD'04), 2004, pp. 128-134.

[79] X. Xie, W. E. Wong, T. Y. Chen, B. Xu. Metamorphic Slice: An Application in SpectrumBased Fault Localization. *Information and Software Technology,* 2013, 55(5): 866-879.

[80] G. Dong, S. Wu, G. Wang, T. Guo, Y. Huang. Security Assurance with Metamorphic Testing and Genetic Algorithm. *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (WI-IAT'10), IEEE Computer Society, 2010, pp. 397-401.

[81] C. Sun, G. Wang, B. Mu. Metamorphic Testing for Web Services: Framework and A Case Study. *Proceedings of the 9th IEEE International Conference on Web Services* (ICWS'11), IEEE Computer Society, 2011, pp. 283-290.

[82] C. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, T. Y. Chen. A Metamorphic Relation-Based Approach to Testing Web Services without Oracles. *International Journal of Web Services Research*, 2012, 9(1): 51-73.

[83] W. K. Chan, S. C. Cheung, K. R. Leung. Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications. *Proceedings of 5th International Conference on Quality Software* (QSIC'05), IEEE Computer Society, 2005, pp. 470-476.

[84] W. K. Chan, S. C. Cheung, K. R. Leung. A Metamorphic Testing Approach for Online Testing of Service-Oriented Software Applications. *International Journal of Web Services Research*, 2007, 4(2): 61-72.

[85] J. Mayer, R. Guderlei. On Random Testing of Image Processing Applications. *Proceedings of the 6th International Conference on Quality Software* (QSIC'06), IEEE Computer Society, 2006, pp. 85-92.

[86] F. C. Kuo, S. Liu, T. Y. Chen. Testing a Binary Space Partitioning Algorithm with Metamorphic Testing. *Proceedings of the ACM Symposium on Applied Computing* (SAC'11), ACM Press, 2011, pp. 1482-1489.

[87] T. H. Tse, S. S. Yau. Testing Context-Sensitive Middleware-Based Software Applications. *Proceedings of the 28th Annual International Computer Software and Applications Conference* (COMPSAC'04). IEEE Computer Society, 2004, pp. 458-466.

[88] W. K. Chan, T. Y. Chen, S. C. Cheung, T. H. Tse, Z. Zhang. Towards the Testing of PowerAware Software Applications for Wireless Sensor Networks. *Proceedings of the 12th AdaEurope International Conference on Reliable Software Technologies* (ICRST'07), Springer, 2007, pp. 84-99.

[89] C. Sun, Z. Wang, G. Wang. A Property-based Testing Framework for Encryption Programs. *Frontiers of Computer Science*, Springer, 2014, 8(3): 478-489.

[90] https://www.dmv.ca.gov/portal/wcm/connect/946b3502-c959-4e3b-b119-91319c27788f/GoogleAutoWaymodisengagereport2016.pdf?MOD=AJPERES, Google Auto Waymo Disengagement Report for Autonomous Driving.

[91]https://www.theatlantic.com/technology/archive/2017/08/insidewaymos-secret-testing-andsimulation-facilities/537648/, Inside Waymo's Secret World for

Training Self-Driving Cars.

[92] I. H. Witten, E. Frank, M. A. Hall, and J. P. Christopher. Data Mining:Practical Machine Learning Tools and Techniques. *Journal of Management Science*, Ubon Ratchathani University, 2005, 3(6): 92-96.

[93] http://www.cleverhans.io/security/privacy/ml/2017/06/14/verifcation.html, The Challenge of Verifcation and Testing of Machine Learning.

[94] K. Pei, Y. Cao, J. Yang, S. Jana. Deepxplore: Automated Whitebox Testing of Deep Learning Systems. *Proceedings of the 26th Symposium on Operating Systems Principles* (SOSP'17), ACM Press, 2017, pp. 1-18.

[95] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, A. Swami. The Limitations of Deep Learning in Adversarial Settings. *Proceedings of the 1st IEEE European Symposium on Security and Privacy* (EuroS&P'16), IEEE Computer Society, 2016, pp. 372-387.

[96] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, A. Criminisi. Measuring Neural Net Robustness with Constraints. *Proceedings of the 10th Annual Conference on Neural Information Processing Systems* (NIPS'16), 2016, pp. 2613-2621.

[97] N. Carlini, D. Wagner. Towards Evaluating the Robustness of Neural Networks. *Proceedings of the IEEE Symposium on Security and Privacy* (SP'17), IEEE Computer Society, 2017, pp. 39-57.

[98] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, A. Criminisi. Measuring Neural Net Robustness with Constraints. *Proceedings of the International Conference on Machine Learning* (ML'17), IEEE Computer Society, 2017, pp. 2613-2621.

[99] S. Gu, L. Rigazio. Towards Deep Neural Network Architectures Robust to Adversarial Examples. *Proceedings of the International Conference on Learning Representations* (ICLR'15), IEEE Computer Society, 2015, pp. 777-780.

[100] X. Huang, M. Kwiatkowska, S. Wang, M. Wu. Safety Verification of Deep Neural Networks. *Proceedings of the International Conference on Computer Aided Verification* (CAV'17), Springer, 2017, pp. 3-29.

[101] K.-Y. Cai. Optimal Software Testing and Adaptive Software Testing in the Context of Software Cybernetics [J]. *Information and Software Technology*, 2002, 44(14): 841–855.

[102] P. E. Ammann and J. C. Knight. Data Diversity: an Approach to Software Fault Tolerance [J]. *IEEE Transactions on Computers*, 1988, 37(4): 418–425.

[103] G. B. Finelli. NASA Software Failure Characterization Experiments [J]. *Reliability Engineering and System Safety*, 1991, 32(1): 155–169.

[104] K.-Y. Cai, H. Hu, and F. Ye. Random Testing with Dynamically Updated Test Profile [C]. *Proceedings of the 20th International Symposium on Software Reliability Engineering* (ISSRE'09), IEEE Computer Society, 2009, pp. 198.

[105] Y. Li, B. B. Yin, J. Lv, and K.-Y. Cai. Approach for Test Profile Optimization in Dynamic Random Testing [C]. *Proceedings of the 39th IEEE Annual International Computer Software and Applications Conference* (COMPSAC'15), IEEE Computer Society, 2015, pp. 466–471.

[106] Z. Yang, B. Yin, J. Lv, K.-Y. Cai, S. S. Yau, and J. Yu. Dynamic Random Testing with Parameter Adjustment [C]. *Proceedings of the 6th IEEE International Workshop on Software Test Automationthe*, Co-located with the *38th IEEE Annual International Computer Software and Applications Conference* (COMPSAC'14), IEEE Computer Society, 2014, pp. 37–42.

[107] J. Lv, H. Hu, and K.-Y. Cai. A Sufficient Condition for Parameters Estimation in Dynamic Random Testing [C]. *Proceedings of the 3rd IEEE International Workshop on Software Test Automationthe*, Co-located with the *35th IEEE Annual International Computer Software and Applications Conference* (COMPSAC'11), IEEE Computer Society, 2011, pp. 19–24.

[108] Barr E T, Harman M, McMinn P, et al. The oracle problem in software testing: A survey[J]. *IEEE Transactions on Software Engineering*, 2015, 41(5): 507-525.

[109] Hernández-Orallo J. Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement[J]. *Artificial Intelligence Review*, 2017, 48(3): 397-447.

2. 项目的研究内容、研究目标，以及拟解决的关键科学问题（此部分为重点阐述内容）；

2.1 研究目标

本课题的研究目标是为开发可靠的智能软件系统提供新型、高效的测试理论，探索智能软件系统的数据驱动式测试方法与技术，开发支持智能软件系统测试的工具原型。

2.2 主要研究内容

在智能软件系统测试领域前沿研究成果基础上，围绕智能软件系统测试中**测试任务描述、测试用例生成、测试结果判定、测试充分性评估、测试平台搭建**等关键问题开展研究，探索智能软件系统的测试框架与优化技术。主要研究内容如下：

（1）智能软件系统的数据驱动式的测试框架

智能软件测试的主要步骤包括测试用例生成、测试执行与测试结果判定。测试用例生成与测试结果判定是智能软件测试领域尚未有效解决的开放问题。智能软件系统要求具有类似人类的智能行为，且大多数智能软件需要和外界进行频繁的交互，导致智能软件系统的输入域异常庞大、系统复杂性较高。显然，穷尽测试无法实现。生成有效的、多方位测试智能软件系统的测试用例是一项具有挑战性的工作。由于智能软件系统的执行逻辑复杂，测试结果判定更加困难，在某些情况下甚至无法确定测试预期。现有的智能软件测试技术利用随机或人工的方式生成测试用例并且主要依靠测试人员验证系统行为，导致测试效率可能不高。另一方面，现有的智能软件测试工作忽略了测试过程对测试效率的影响。在测试资源有限的情况下，利用数据驱动的方式（基于测试的历史信息），控制测试过程，提高测试效率。研究智能软件系统的数据驱动式测试时，首先需要建立面向智能软件系统的数据驱动式测试框架。研究如下问题：

- **智能软件系统的故障特点分析：**智能软件系统的故障集中出现在：数据预处理和决策模块的代码实现部分以及决策模块的逻辑部分。实现数据处理和决策模块的程序语言不同，相应地，故障类型与特点也不同。另外，决策模块的逻辑是从数据中习得且呈现出异常复杂的特点，与传统代码故障差异较大。因此，与决策模块逻辑相关的故障难以理解。
- **智能软件系统的故障检测机制：**测试用例生成是智能软件系统测试的关键步骤。在测试过程中，为了增加发现系统行为异常的概率，生成的测试用例应当考虑系统的内部结构。智能软件系统的故障检测不仅要关注测试用例的生成方式，还要考虑如何尽可能地覆盖不同的执行逻辑。

（2）基于场景的智能软件系统测试任务描述方法

针对智能软件系统测试任务难以准确描述的问题，研究基于场景的智能软件系统的测试任务描述方法。分析智能软件系统应用场景及任务中与软件质量相关的特征，通过这些特征分析智能软件系统在运行时所应满足的特性，针对这些特性为智能软件系统制定测试任务。研究如下内容：

- **智能软件系统应用场景及任务的软件质量相关特征提取及描述机理：**智能软件系统运行时所体现的功能往往是适应特定场景及任务的。因此，描述智能软件系统的待测功能需要从其应用场景和任务入手。不同的场景及任务具有不同的特征或属性，需要提取与软件质量相关的特征。研究智能软件系统应用场景和任务的软件质量相关特征提取与描述机理，为分析智能软件系统的运行时特性提供基础。
- **场景特征所关联的智能软件系统运行时特性分析机理：**针对某一场景或任务下的特征，研究如何分析智能软件系统为适应或满足这一特征而必须满足的运行时特性。

  基于场景特征的智能软件系统测试任务描述方法：在得到智能软件系统为适应或满足场景特征而必须满足的运行时特性后，研究如何依据运行时特性描述并指定智能软件系统测试任务的方法。

（3）智能软件系统的高效、多样性测试用例生成与优化

针对现有智能软件系统测试用例生成技术的诸多局限性，**考虑智能软件系统的特点，依据现有测试用例生成技术与相关理论，研究新型数据驱动的智能软件系统的测试用例生成技术。**在此基础上，研究测试用例的优化技术，包括：测试用例生成与选择，测试用例执行与测试用例优先级排序等。需要研究以下内容：

- **在测试用例生成与选择方面：**智能软件系统输入参数的多源性导致输入空间巨大。增加测试用例集的多样性有助于更有效地检测智能软件系统的异常行为。为了实现测试用例集多样性，探索：（a）满足某种覆盖准则的测试用例生成方法；（b）结合场景与功能的测试用例生成技术，将场景测试数据与功能测试数据相结合来实现智能软件系统自顶向下的测试；（c）基于参数组合优化的测试生成技术，实现减少测试用例数目的同时覆盖尽可能多的组合。
- **在测试用例执行方面：**在生成的测试用例集基础上，利用历史数据、实时数据等控制测试的过程，选择下一个测试用例，进一步提高智能软件故障被揭示的效率。
- **在测试用例优先级排序方面：**通过改变测试用例的执行顺序以提早执行到揭

示故障的测试用例。为此，即将执行的测试用例应尽可能与已经执行的测试用例不同。通过测试用例距离度量的方式选择最不同的测试用例，并以此为基础提出基于距离的智能软件系统测试用例优先级排序技术。

（4）测试充分性以及测试用例集充分性的评估指标

智能软件系统获取决策逻辑的方式使得如何评估被测对象的测试充分性以及测试用例集的充分程度是一个重要问题。传统软件测试中的测试充分性指标（语句覆盖、分支覆盖和 MC/DC 覆盖等）不能完全适用于智能软件测试。原因是，在智能软件测试中，不仅要考虑数据处理模块、决策模块等对应的代码是否正确实现，还需要考虑影响决策逻辑的内部因素（神经元、激活函数和阈值等）。另一方面，由于智能软件获取逻辑的方式，传统软件测试中评估测试用例集充分性的技术（变异分析和各种覆盖准则）也不能完全适用于智能软件系统。研究一下问题：

- **提出测试充分性的评估指标：**传统软件的测试充分性度量指标可以评估智能软件中代码部分的测试充分性，不能对决策模型进行有效的评估。为此，借鉴传统软件测试充分性评估指标的提出思路，并结合决策模型的特点，提出适用于智能软件的测试充分性指标。
- **提出测试用例集充分性的评估指标：**变异分析和覆盖准则可以评估传统软件测试用例集的充分性，但由于缺少针对智能软件的变异算子，变异分析技术不能直接运用到智能软件测试中。为了解决这一问题，分析以往的测试历史，提取故障模式，总结面向智能软件的变异算子。

（5）智能软件系统执行结果的判定方式

智能软件系统的测试工作主要通过测试人员判断系统行为是否正确，这种方式不仅需要大量的测试资源，还易出现误判的情况（将错误的行为判断为正确的行为）。因此，提出自动化验证系统行为的方法，可以极大提高测试效率与准确度。研究以下问题：

- **提出一种自动或者直观的方式判断系统行为是否正确：** 人工验证系统行为的方式不仅需要占用大量的测试资源而且不能保证判断的准确度。因此，提出一种自动地验证系统行为的方式是有必要的。传统软件测试中缓解测试预期问题的技术有很多（例如：N 版本、断言等），其中一些技术运用到智能软件测试效果可能不好。例如，断言技术需要在代码中插入一些可以获取执行信息的代码，但很多智能软件系统的执行逻辑不是由程序控制的。考虑将 N 版本以及交叉验证的的方式作为验证系统行为的机制。
- **基于蜕变测试的智能软件系统的测试结果验证技术：**蜕变测试是一种无需测试预期的软件测试技术，在待测软件的测试预期不存在的情况下也能对其进

行有效测试。该测试技术通过判断待测软件的多个测试用例之间是否满足一些必要的属性来测试程序。这些必要的属性被称为蜕变关系，隐含于待测软件的功能规格说明中，是验证测试结果及判断待测软件是否满足特定的功能需求。研究基于蜕变测试的智能软件系统测试结果验证技术，需要研究如下内容：**（a）面向智能软件系统的蜕变关系识别方法：**智能软件系统中隐含的蜕变关系是验证测试结果的关键。如何从智能软件系统中识别蜕变关系是一个重要的问题。研究基于范畴划分方法的和基于数据变异的智能软件系统蜕变关系识别方法；**(b)基于蜕变测试的智能软件系统测试结果验证实现机理：**通过判断蜕变关系所涉及测试用例的输出结果之间是否满足蜕变关系，实现智能软件系统测试结果的验证，判断待测软件是否满足功能需求。

（6）智能软件系统的模拟测试平台

针对真实环境下的智能软件系统测试存在高昂且难以负担的成本的问题，搭建智能软件系统的模拟测试平台。通过模拟智能软件系统与外界环境的交互，实现在接近真实的环境下对智能软件系统展开测试。需要研究以下内容：

- **数据驱动的智能软件系统应用环境建模：**针对不同类型的智能软件系统，分析软件系统在主要应用场景下的真实历史运行数据，细化应用场景下的事物、交互对象、交互方式，并以此为基础构建智能软件系统在各应用场景下的模型。
- **智能软件系统应用环境模拟的实现机理：**以智能软件系统的应用场景模型为指导，模拟系统实际应用环境下的各种事物及对象，较为真实地构建智能软件系统的实际应用环境，为智能软件系统的测试奠定基础。

（7）智能软件系统与环境实时交互的可靠性验证

很多智能软件系统通过各种传感器获取外界环境信息，将环境信息作为输入传递给决策模块，然后决策模块根据内部逻辑控制智能软件系统的行为。准确地辨别、分析外界事物是智能软件系统做出正确决策的前提。如果不能准确识别可能带来安全隐患的事物，智能软件系统行为的正确性就得不到保证。研究如下问题：

- **智能软件系统识别外界事物的可靠性：** 在实际环境中，很多事物易于和周围环境混淆。 例如，无人车主要通过摄像机获取路况和周围车辆等关键信息的二维图像，这种信息呈现方式使得易于环境混淆的事物更加难以区别。如果无人车不能区分这些事物与周围环境的差别，就可能出现事故。
- **智能软件系统将外界信息转化为决策模型输入的准确性：**智能软件系统将获取的外界信息进一步加工得到决策模块可以直接使用的数据。在这个过程中，外界信息与处理后的信息的等价性难以保证。

（8）面向智能软件系统的质量评价指标及模型

针对智能软件系统的质量评价指标及模型存在缺失的问题，研究面向智能软件系统的质量评价指标及模型。针对智能化软件系统呈现出的独有特点，提出新型系统智能质量评价指标及模型。研究面向智能软件系统的质量评价指标及模型，需要研究以下内容：

- **面向系统智能的质量评价指标及模型：**重点分析智能软件系统智慧程度的影响因素，提炼用于描述软件系统智能程度的软件质量特征，提出评估软件系统智能成熟度的质量评价指标，并在传统软件质量模型的基础上构建新型面向系统智能的质量评估模型。
- **多目标情景下的智能软件系统质量评估指标及模型：**分析多目标情景智能软件系统满足用户需求程度的描述性软件质量特征，提出多目标情景下的智能软件系统质量评估指标，并以传统软件质量模型为参考提出新型多目标情景下的软件质量评估模型。

（9）面向智能软件系统的测试支持工具与实例验证

- 设计与实现一个面向智能软件系统的测试支持工具。
- 采用多个智能软件系统及应用场景验证面向智能软件系统的数据驱动式测试方法与技术的有效性与效率。

3. 拟采取的研究方案及可行性分析（包括研究方法、技术路线、实验手段、关键技术等说明）；

3.1 研究方法

本课题将结合软件测试领域的最新研究成果，**探索有效且高效的面向智能化软件系统的新型软件测试技术，重点研究数据驱动的智能软件系统的测试用例生成与优化，面向智能软件系统的测试结果判定**，包括数据驱动的智能软件系统测试用例生成实现机理、基于组合优化的智能软件系统测试用例选择实现机理、基于距离的智能软件系统测试用例优先级排序实现机理、基于覆盖准则的智能软件系统测试用例集充分性评估、面向智能软件系统的蜕变关系识别方法、基于蜕变测试的智能软件系统测试结果验证实现机理。**本课题将以智能化软件系统的典型实例—无人驾驶系统作为研究对象，通过实验的方式验证所提方法及技术的有效性与效率**。

3.2 技术路线、关键技术及实验手段

针对智能软件系统的**学习深层知识、融合跨界数据、数据处理层级化、决策逻辑不受控、系统输出难以验证、集成群体智慧**等特点带来的问题，本课题探索数据驱动式测试方法与技术，包括：面向智能软件系统的测试框架、面向智能软件系统的待测功能描述方法、智能软件系统的模拟测试平台、数据驱动的智能软件系统的测试用例生成与优化技术、面向智能软件系统的测试结果验证技术、面向智能软件系统的质量评价指标及模型。拟采取的技术路线与方法如下：

（1） **构建面向智能软件的数据驱动式测试框架。**具体来说：（a）在智能软件系统的故障检测方面，根据智能软件系统的特点，智能软件的故障分为：代码故障和决策模型的故障（不能根据输入表现出正确的行为）。针对上述智能软件的故障，执行的测试用例应能够满足针对代码的不同类型的测试充分性指标，还需要全方位地测试决策模型；（b）在构建智能软件的数据驱动式测试框架时，首先需要生成具有上述特点并满足某种测试用例集充分性度量指标的测试用例集，然后利用历史或者实时的测试数据来选取下一个测试用例，最后判断智能软件系统的行为是否正确。

（2） **通过智能软件系统的历史数据及未来可能的应用环境，分析并归纳待测功能：**针对分析得到的应用场景及任务，结合相关领域知识，重点探究从场景及任务中提取出哪些软件质量相关的特征，以及如何从场景及任务中提取出这些特征。在得到这些与软件质量相关的特征后，需要分析智能软件系统为适应或满足这一特征而必须要满足的运行时特性。针对这一研究内容，拟采取人工的方式为这些特征设定智能软件系统所必须满足的运行

时特性。此时，在指定的应用场景与任务下，测试人员能够依据场景或任务的特征关联出智能软件系统应满足的所有运行时特性。测试人员可依据这些运行时特性为智能软件以系统且有效地方式制定测试任务。

（3） **以开源智能软件仿真平台为基础，构建测试仿真平台。具体来说：**（a）无人驾驶系统的历史运行数据中包含着无人驾驶汽车的传感器对外界环境的实时感知结果。通过对这些感知结果进行分析得到无人驾驶系统在不同应用场景及任务下所可能涉及的事物、交互对象及交互方式，构建智能软件系统各个应用场景的模型;（b）修改开源的无人驾驶仿真平台源代码（例如 Carla、AirSim、Apollo 和 Autoware），添加软件测试相关的必要功能模块，构建无人驾驶系统的模拟测试平台。

（4） **以大量、多类型（历史、实时、开发与维护等）数据为基础，提出**智能**软件系统的测试用例生成与优化技术。**具体来说：基于场景的测试用例生成技术关注于系统层面的测试，而基于功能的测试用例生成技术更加关注于系统的独立模块，考虑将两种技术进行优势互补，实现场景到功能的贯通，具体来说：针对不同的应用场景及任务，分析无人驾驶系统在运行时所涉及的功能模块，以此建立场景及任务与系统功能的映射关系；在此基础上，可以通过使用软件系统每个独立模块的测试结果来度量智能软件系统在给定应用场景及任务下的功能质量，实现面向应用场景的智能软件系统质量度量的量化。测试用例优化关键问题的解决思路：（a）对于智能化软件系统的测试用例选择，通过**组合优化技术**在测试用例集中选择满足一定组合覆盖条件的测试用例子集，代替原有测试用例全集；研究满足不同组合覆盖条件测试用例集在故障检测能力方面的差异，找出在一定条件下具有最优故障检测能力的组合覆盖条件；（b）对于智能化软件系统的测试用例排序，通过借鉴**适应性随机测试的思想**，每次在测试用例全集中选择与已执行测试用例最远的测试用例；提出新的距离度量指标量化输入之间的距离并且研究不同的距离度量方法对排序后测试用例集故障检出速度的影响；（c）对智能软件的测试用例执行，考虑将控制理论引入智能软件测试中，**利用测试的历史信息**，控制下一个测试用例的选择，达到尽快揭示智能软件故障的目的；（d）对于智能化软件系统测试充分性及测试用例集的评估，从**传统的覆盖准则与神经网络的角度**，提出新的覆盖准则，基于提出的新准则判断智能软件的测试程度以及测试用例集的充分性。另外，从变异测试的角度，结合**智能软件系统的故障模式**，提出针对智能软件的变异算子，进一步评估测试用例集的测试充分性；

（5） **在传统缓解测试预期问题的技术上，优化或提出适用于智能软件测试预**

期问题的技术。具体来说：（a）**以蜕变关系为基础实现智能软件系统测试结果验证，**具体方法是判断智能软件系统的多个测试用例的实际输出结果之间是否满足蜕变关系，若不满足，则表明待测系统应满足的必要属性被违反，即系统潜藏故障；（b）研究现有蜕变关系识别方法是否适用于智能化软件系统；使用变异分析技术**研究不同蜕变关系识别技术识别得到的蜕变关系集的故障检测能力**，为智能化软件系统的蜕变关系选择提供启发式规则；（c）利用**交叉引用的方式**判断智能软件系统的行为是否符合预期，具体方法是将同一输入在不同公司开发的无人驾驶汽车上执行，观察无人驾驶汽车的行为，若存在一辆或者若干辆无人驾驶汽车的行为与其它不一致，表明其中的某些无人驾驶汽车存在故障；（d）结合传统的 N 版本技术思想，将同一输入在智能软件系统的不同版本中执行，观察每一个系统的行为，若存在一个或者若干个智能软件系统的行为与其它不一致，表明某些版本存在故障。

（6） **基于智能软件的质量特征与应用场景，提出适用于智能化软件系统的质量评价指标。**关键问题的解决思路：以前文提到的智能化软件应用场景及任务的质量特征提取方法为基础，通过分析大量的应用场景及任务，得到多维度多方面的质量特征；以这些质量特征为基础提出面向智能化软件系统的软件质量评价指标，建立指标与指标之间的层次结构关系，最终形成面向智能化软件系统的质量评价模型。

（7） **采用经验研究、实例研究与变异分析的方式，以开源系统为研究对象，评估本文所提方法与技术的有效性与效率。**具体来说：（a）采用变异分析技术为开源无人驾驶系统产生变异体集合；（b）应用提出的测试用例生成技术为开源无人驾驶系统构建适用的测试用例集，使用所提测试用例选择及优先级排序技术进一步对测试用例集进行精简与排序，通过充分性评估技术度量测试用例集的充分性以及智能软件系统的测试充分性；（c）在测试预期难以获取的情形下，通过所提测试结果验证技术来判断开源无人驾驶系统的行为是否符合预期；（d）使用测试用例集检测开源无人驾驶系统的变异体，使用测试用例对应的预期输出或使用其它判定测试结果技术，统计与评估故障检测的有效性与效率；（e）通过所提智能软件系统的质量评价指标及模型度量开源无人驾驶系统满足给定需求或特性的程度。

4. **本项目的特色与创新之处；**

5. **年度研究计划及预期研究结果**（包括拟组织的重要学术交流活动、国际合作与交流计划等）。

（二）研究基础与工作条件

1. **研究基础**（与本项目相关的研究工作积累和已取得的研究工作成绩）；

2. **工作条件**（包括已具备的实验条件，尚缺少的实验条件和拟解决的途径，包括利用国家实验室、国家重点实验室和部门重点实验室等研究基地的计划与落实情况）；

3. **正在承担的与本项目相关的科研项目情况**（申请人和项目组主要参与者正在承担的与本项目相关的科研项目情况，包括国家自然科学基金的项目和国家其他科技计划项目，要注明项目的名称和编号、经费来源、起止年月、与本项目的关系及负责的内容等）；

4. **完成国家自然科学基金项目情况**（对申请人负责的前一个已结题科学基金项目（项目名称及批准号）完成情况、后续研究进展及与本申请项目的关系加以详细说明。另附该已结题项目研究工作总结摘要（限 500 字）和相关成果的详细目录）。

（三）其他需要说明的问题

1. 申请人同年申请不同类型的国家自然科学基金项目情况（列明同年申请的其他项目的项目类型、项目名称信息，并说明与本项目之间的区别与联系）。

2. 具有高级专业技术职务（职称）的申请人或者主要参与者是

否存在同年申请或者参与申请国家自然科学基金项目的单位不一致的情况；如存在上述情况，列明所涉及人员的姓名，申请或参与申请的其他项目的项目类型、项目名称、单位名称、上述人员在该项目中是申请人还是参与者，并说明单位不一致原因。

3. 具有高级专业技术职务（职称）的申请人或者主要参与者是否存在与正在承担的国家自然科学基金项目的单位不一致的情况；如存在上述情况，列明所涉及人员的姓名，正在承担项目的批准号、项目类型、项目名称、单位名称、起止年月，并说明单位不一致原因。

4. 其他。