



本科毕业设计 (论文)



智能驾驶危险仿真场景的生成

题 目: 和优化算法

学生姓名: 刘东林

学 号: 2123020065

专 业: 人工智能

班 级: 人工智能 2102 班

指导老师: 王海东教授

计算机学院

2025 年 5 月

湖南工商大学本科毕业设计诚信声明

本人郑重声明：所呈交的本科毕业设计 湖南工商大学学位论文 L^AT_EX 模板使用示例 v0.1 是本人在指导老师的指导下，独立进行研究工作所取得的成果，成果不存在知识产权争议，除文中已经注明引用的内容外，本设计不含任何其他个人或集体已经发表或撰写过的作品成果。对本设计做出重要贡献的个人和集体均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名:刘东林

日 期: 2025 年 1 月 3 日

摘要

在智能驾驶系统的发展过程中，仿真场景的生成与优化是确保其安全性和可靠性的重要手段。本项目综述了智能驾驶危险仿真场景的生成和优化技术。首先，基于自然驾驶数据，识别并提取出具有代表性的危险驾驶场景，为仿真场景的构建提供了数据基础。其次，通过多维场景自动提取和融合方法，识别出典型的行车场景，如巡线、跟车、邻车切入等，并将其与动态驾驶场景进行融合，以生成更为复杂和真实的测试场景。此外，针对现有测试场景中危险场景数量不足的问题，提出了一种基于聚类分析和重要性采样的测试用例生成和增强方法，有效提高了危险场景的测试覆盖率和测试效率。最后，开发了一种自动化仿真测试平台，实现了测试场景的快速构建、仿真软件的联合调用以及结果分析与报告生成的全自动化执行。通过这些方法，能够显著提高智能驾驶系统在仿真环境中的安全测试效率，为智能驾驶技术的进一步发展提供了有力支持。

关键词：智能驾驶 仿真场景 危险场景生成 自动化技术

ABSTRACT

In the development of intelligent driving systems, the generation and optimization of simulation scenarios are crucial for ensuring their safety and reliability. This project reviews the technologies for generating and optimizing dangerous simulation scenarios in intelligent driving. Firstly, based on natural driving data, representative dangerous driving scenarios are identified and extracted, providing a data foundation for the construction of simulation scenarios. Secondly, through multi-dimensional scenario automatic extraction and fusion methods, typical driving scenarios such as line-following, following, and lane-changing are identified and integrated with dynamic driving scenarios to generate more complex and realistic test scenarios. Additionally, to address the issue of insufficient dangerous scenarios in existing test scenarios, a test case generation and enhancement method based on cluster analysis and importance sampling is proposed, which effectively improves the test coverage and efficiency of dangerous scenarios. Finally, an automated simulation testing platform has been developed, enabling the rapid construction of test scenarios, joint invocation of simulation software, and fully automated execution of result analysis and report generation. These methods can significantly improve the safety testing efficiency of intelligent driving systems in simulation environments, providing strong support for the further development of intelligent driving technology.

Key words: Intelligent Driving Simulation Scenarios Dangerous Scenario Generation Automation Technology

目录

第 1 章 引言	1
1.1 研究背景	1
1.2 研究目的和意义	1
1.2.1 研究目的	1
1.2.2 意义	2
1.3 国内外研究现状	2
1.4 研究方法	2
1.5 创新点	5
1.6 自动驾驶危险场景生成	6
第 2 章 危险仿真场景生成实验研究准备	10
2.1 实验准备与数据采集	10
2.1.1 实验准备	10
2.1.2 数据采集	10
2.2 危险仿真场景提取的特征参数确定	10
2.2.1 场景提取参数确定	11
2.2.2 评估指标	12
2.3 讨论与改进方向	13
第 3 章 实验流程	14
3.1 ChatScene 框架设计	14
3.1.1 流程	14
3.2 ASIL-Gen 场景优化与分类	16
3.3 讨论与未来工作	17
3.3.1 优势与局限	17
3.3.2 未来方向	17
第 4 章 技术实现	20
4.0.1 技术实现要素	20
4.1 详细操作	21
4.2 更多技术支持	25
4.3 总结与展望	28
4.3.1 总结	28
4.3.2 展望	30

4.4 参考文献	32
致谢	35
附录 A 附录代码	37
A.1 chatscene	37
A.2 ASIL-Gen	39
A.3 scene_runner	41
A.4 scene_runner	47
参考文献	36

第1章 引言

随着科技的飞速发展，智能驾驶技术逐渐从实验室走向现实道路，成为汽车工业和人工智能领域的重要研究方向。智能驾驶系统通过集成先进的传感器、控制器、执行器以及复杂的算法，实现对车辆的自主控制和决策，旨在提高道路安全性、缓解交通拥堵、降低能源消耗以及提升驾驶舒适性。然而，智能驾驶系统的安全性和可靠性始终是其发展的关键瓶颈。在复杂的道路交通环境中，智能驾驶系统需要面对各种突发状况和潜在危险，如行人突然横穿马路、车辆紧急变道、恶劣天气条件等，这些场景对系统的感知、决策和控制能力提出了极高的要求。

1.1 研究背景

为了在智能驾驶系统投入实际应用之前充分验证其性能，传统的道路测试方法存在诸多局限性。道路测试周期长、成本高，且难以覆盖所有可能的危险场景，特别是那些发生概率较低但后果严重的事故场景。此外，道路测试还存在一定的安全风险，可能对测试人员和周围环境造成威胁。因此，仿真技术应运而生，成为智能驾驶系统测试和验证的重要手段。通过构建虚拟的仿真场景，可以在计算机中模拟真实的道路交通环境，让智能驾驶系统在虚拟世界中“驾驶”，从而在可控、低成本的条件下，全面、高效地评估系统的性能。

在仿真场景的构建过程中，危险场景的生成和优化尤为关键。危险场景是指那些可能导致智能驾驶系统出现错误决策、控制失效或者发生事故的场景。准确地生成和优化这些危险场景，可以帮助研究人员深入理解智能驾驶系统在面对危险时的行为表现，发现系统潜在的缺陷和不足，进而针对性地改进系统的设计和算法，提高系统的鲁棒性和安全性。然而，危险场景的生成并非易事，需要综合考虑道路交通规则、车辆动力学特性、传感器感知范围、环境因素等多种因素，且要保证生成的场景具有高度的真实性和代表性，能够真实反映智能驾驶系统在实际道路中可能遇到的危险情况。

1.2 研究目的和意义

1.2.1 研究目的

本研究旨在深入探讨智能驾驶危险仿真场景的生成和优化技术，以期为智能驾驶系统的安全测试和性能提升提供有力支持。研究目的的具体包括：

系统地梳理和总结现有的智能驾驶危险仿真场景生成方法，分析其优缺点和适用范围，为后续的研究工作奠定理论基础。研究和开发新的危险场景生成算法，能够更高效、更准确地生成多样化的危险场景，提高仿真测试的覆盖率和有效性。探索危险场景的优化策略，通过优化场景的参数设置、布局结构等，使生成的场景更具挑战性和针对性，能够更好地测试智能驾驶系统的极限性能。构建一个完整的智能驾驶危险仿真场景生成

和优化框架，整合数据采集、场景生成、场景优化、仿真测试等环节，为智能驾驶系统的开发和测试提供一套实用的工具和流程。本研究的意义主要体现在以下几个方面：

1.2.2 意义

理论意义：丰富和完善智能驾驶仿真测试领域的理论体系，为智能驾驶危险场景的生成和优化提供新的思路和方法，推动相关学术研究的深入发展。

技术意义：提高智能驾驶仿真测试技术的水平，为智能驾驶系统的开发和测试提供更高效、更可靠的工具，促进智能驾驶技术的成熟和应用。

实际应用意义：通过生成和优化危险仿真场景，能够更全面地评估智能驾驶系统的安全性，提前发现和解决潜在的安全问题，降低智能驾驶系统在实际应用中的风险，为智能驾驶车辆的上路行驶提供安全保障，具有重要的社会价值和经济价值。

1.3 国内外研究现状

自动驾驶测试方法：

自动驾驶汽车初期功能单一，如自动紧急制动、自适应巡航等。工程师设计了基于功能的低等级自动驾驶测试场景，如欧盟新车安全评鉴协会和国际标准化组织的测试场景。这种方法可重复性强，效率高，但应用于高等级自动驾驶时存在不足：只能测试单一功能，无法验证功能综合表现，且难以体现系统自主决策能力。

随着技术发展，自动驾驶汽车成为人-车-路-环境-任务的强耦合系统，测试内容涵盖感知、建图定位、决策规划和控制执行等模块。测试不仅要在软件层面进行，还需在真实道路环境如封闭测试场地和开放道路进行。国外有密歇根 Mcity、瑞典 AstaZero、英国 Mira City Circuit 等知名封闭测试场。国内约有 50 个封闭测试场，其中 30 个具备高等级自动驾驶测试能力。2018 年，交通运输部认定了北京通州、重庆车检院、长安大学等 3 家测试基地。但封闭测试场地建设成本高、测试成本昂贵、场景有限，缺乏统一测试标准和规范，难以满足自动驾驶测试要求。自动驾驶仿真技术日益成熟，受到汽车厂商关注。虚拟仿真测试灵活、高效、可重复、安全且有针对性，可生成丰富测试场景替代实车路测。主流仿真测试平台包括自动驾驶仿真引擎、测试场景库和测评体系。西安深信科创、北京 51Sim、清华大学苏州汽车研究院、中国汽车研究中心、SIEMENS 公司等推出了不同仿真测试平台，集成了法规工况、交通事故数据库等场景库。英伟达 Omniverse Cloud API 通过传感器仿真加速性能测试，连接自动驾驶数字试验场。国内外仿真测试平台对比详见表 3。自定义测试场景是自动驾驶安全测试的关键，但现有场景库量少质弱，难以全面测试系统。建立大规模测试场景库是虚拟仿真技术的核心问题，对评估算法、提升测试效率、加速量产至关重要。

1.4 研究方法

数据驱动分析

交通事故数据分析：收集并整理大量的交通事故案例数据，涵盖不同类型的事故（如追尾、侧撞、翻车等）以及事故发生的各种因素（如天气条件、道路状况、车辆类型、驾驶行为等）。运用数据挖掘技术，如关联规则挖掘、聚类分析等，从海量数据中挖掘出事故发生的潜在规律和关键影响因素，识别出常见的危险场景模式。例如，通过分析发现，在雨天湿滑路面上，车辆在高速行驶时容易发生侧滑，进而引发侧撞事故；或者在交通拥堵路段，车辆频繁变道、加塞等行为容易导致追尾事故等。

驾驶行为数据挖掘：采集大量的驾驶行为数据，包括车辆的行驶轨迹、速度、加速度、转向角度、刹车力度等，以及驾驶员的操作行为（如踩油门、踩刹车、打方向盘等）。利用机器学习算法，如随机森林、支持向量机等，对驾驶行为数据进行分类和预测，识别出危险驾驶行为模式。例如，通过分析发现，驾驶员在高速行驶时突然急刹车、急转弯等行为容易导致车辆失控，进而引发事故；或者在跟车过程中，车距过近且跟车速度过快，容易发生追尾事故等。

危险场景类别定义与参数设置

场景分类框架构建：基于数据驱动分析的结果，构建一个系统化的危险场景分类框架。将危险场景按照不同的维度进行分类，如按照事故类型（追尾、侧撞、翻车等）、道路类型（高速公路、城市道路、乡村道路等）、天气条件（晴天、雨天、雪天等）、交通状况（拥堵、畅通等）等。为每一类危险场景设定具体的参数和条件，如车辆的速度范围、加速度范围、转向角度范围、车距范围等。例如，在高速公路追尾场景中，设定车辆的初始速度为 80-120km/h，车距为 10-20 米，加速度为 $-5-0m/s^2$ 等。

仿真环境参数配置：使用 CARLA 等仿真软件，根据定义的危险场景类别和参数，设置仿真环境中的各种参数。包括天气参数（如雨量大小、雪量大小、雾浓度等）、光照参数（如太阳高度角、光照强度等）、道路参数（如路面材质、摩擦系数等）、交通密度参数（如车辆数量、行人数量等），以模拟真实世界中的各种复杂条件。例如，在模拟雨天追尾场景时，设置雨量为中雨，路面摩擦系数降低 20

手动控制场景实现

Python 脚本编写：利用 Python 语言编写脚本，控制仿真环境中的车辆和行人行为。通过脚本设定车辆的初始位置、速度、加速度、转向角度等参数，以及行人的行走路径、速度等参数。例如，编写一个脚本，让车辆在高速公路的某一车道上以 100km/h 的速度行驶，然后在某一时刻突然急刹车，同时让另一辆车以更高的速度从后方驶来，模拟追尾场景。

初始状态配置：在仿真开始前，配置好场景的初始状态。包括车辆的初始位置、速度、方向等，以及行人的初始位置、速度等。确保场景的初始状态符合设定的危险场景类别和参数。例如，在模拟行人横穿马路场景时，将行人初始位置设定在道路一侧的人行道上，速度设定为正常步行速度，车辆初始位置设定在距离行人一定距离的道路上，速度设定为正常行驶速度。

强化学习算法应用

DQN 算法选择与实现：选择 DQN (Deep Q-Network) 强化学习算法，用于在 CARLA

环境中训练和测试模型，生成对抗性场景。DQN 算法通过深度神经网络来近似 Q 值函数，能够处理高维的状态空间和动作空间，适用于复杂的仿真环境。实现 DQN 算法时，需要设计合适的状态表示、动作空间、奖励函数等。例如，状态表示可以包括车辆的速度、位置、加速度、周围环境信息等；动作空间可以包括车辆的加速度、转向角度等；奖励函数可以设定为避免碰撞、保持安全距离等。

对抗性场景生成：在 CARLA 环境中，利用 DQN 算法训练智能体，使其能够生成对自动驾驶系统构成挑战的对抗性场景。智能体通过与环境的交互，学习如何在复杂场景中做出最佳决策，生成能够有效测试自动驾驶系统性能的场景。例如，智能体可以学习如何在交通拥堵路段中，通过频繁变道、加塞等行为，制造出复杂的交通环境，考验自动驾驶系统的决策和控制能力。

数据收集与效果量化

传感器数据采集：在仿真过程中，收集车辆的传感器数据，包括速度、位置、加速度、转向角度、刹车力度等。这些数据可以通过仿真软件提供的 API 接口获取。例如，在 CARLA 中，可以使用车辆的传感器组件来获取车辆的实时数据。

事件记录：记录仿真中的关键事件，如碰撞、紧急制动、车道偏离等。这些事件可以通过仿真软件的事件监听机制来实现。例如，在 CARLA 中，可以使用事件监听器来监听车辆的碰撞事件，并记录碰撞发生的时间、位置、速度等信息。

评价指标设定：设定量化仿真效果的评价指标，如碰撞率、反应时间、安全距离保持率等。这些指标可以用于评估生成的危险场景对自动驾驶系统的影响。例如，碰撞率可以表示为在一定仿真时间内，发生碰撞的次数与总仿真次数的比值；反应时间可以表示为自动驾驶系统从检测到危险到做出反应的时间间隔。

数据分析：使用统计方法对收集到的数据进行分析，评估场景生成的效果。例如，通过计算碰撞率的平均值、标准差等统计量，分析不同危险场景对自动驾驶系统的影响程度；或者通过绘制反应时间的分布图，了解自动驾驶系统在不同场景下的反应速度。

结果展示

图表展示：使用数据可视化工具（如 Matplotlib、Seaborn）制作图表展示仿真结果。例如，绘制碰撞率随仿真时间变化的折线图，或者绘制反应时间的直方图，直观地展示仿真结果。

视频展示：剪辑仿真视频，直观展示场景和车辆行为。例如，将仿真过程中生成的对抗性场景录制为视频，展示车辆在复杂交通环境中的行驶情况，以及自动驾驶系统在面对危险时的决策和控制过程。

场景实现与验证

API 应用：使用 CARLA API 实现场景脚本开发，自动化场景生成过程。通过 API 调用，可以实现对车辆、行人、环境等的控制，以及对仿真过程的监控和数据采集。

仿真测试：在仿真环境中测试场景，验证其合理性和挑战性。通过多次仿真测试，观察场景的运行情况，检查是否存在不合理或不符合预期的情况。根据测试结果调整场景参数，优化场景设计。例如，在测试行人横穿马路场景时，观察行人是否能够安全地

通过马路，自动驾驶系统是否能够及时做出避让反应等。

仿真场景构建技术研究

研究综述：对场景自动构建方法进行研究综述，总结目前存在的各种方法，如基于规则的方法、基于数据驱动的方法、基于模型的方法等，分析其优缺点和适用范围。

算法设计：设计基于复杂度组合理论的测试场景生成算法。该算法通过组合不同的场景元素（如车辆、行人、障碍物等）和参数（如速度、位置等），生成具有不同复杂度的测试场景。在 CARLA 环境中实现和测试算法，验证其有效性。

仿真测试技术研究

评价方法体系建立：建立智能驾驶系统测试评价方法体系，设计测试流程。该体系包括测试目标的设定、测试用例的选择、测试过程的执行、测试结果的分析等环节。

测试平台搭建：搭建仿真场景测试平台，执行大规模自动化测试。该平台可以实现对多个测试用例的自动执行、数据采集和结果分析，提高测试效率和准确性。

1.5 创新点

多源数据融合的场景生成方法

融合多种数据源：本研究提出了一种融合多源数据的危险场景生成方法，综合考虑了车辆行驶数据、交通流量数据、环境感知数据等。通过融合这些数据，能够更全面地了解道路交通环境，生成更为真实和多样化的危险场景。例如，结合车辆行驶数据和交通流量数据，可以模拟出在交通拥堵路段中，车辆频繁变道、加塞等行为引发的复杂交通场景；结合环境感知数据，可以模拟出在恶劣天气条件下，车辆行驶的困难情况，如雨天路面湿滑导致的车辆失控等。

提高场景的真实性：多源数据融合方法能够提高生成场景的真实性。传统的单一数据源生成方法可能存在数据局限性，无法全面反映道路交通环境的复杂性。而多源数据融合方法通过整合不同来源的数据，能够弥补单一数据源的不足，生成的场景更加接近真实世界中的交通情况，为自动驾驶系统的测试提供了更可靠的依据。

基于强化学习的场景优化算法

动态调整场景参数：本研究设计了一种基于强化学习的危险场景优化算法，通过智能体与仿真环境的交互学习，动态调整场景参数。智能体在仿真环境中不断尝试不同的场景参数组合，根据奖励函数的反馈，学习到最优的场景参数设置，使生成的场景更具挑战性和针对性。例如，在模拟车辆超车场景时，智能体可以动态调整前车的速度、加速度以及两车间的距离等参数，生成更加危险的超车场景，有效测试自动驾驶系统的决策和控制能力。

提高场景的挑战性：强化学习算法能够提高生成场景的挑战性。传统的场景生成方法可能生成的场景较为简单，难以充分测试自动驾驶系统的极限性能。而强化学习算法通过不断优化场景参数，能够生成更加复杂和危险的场景，逼迫自动驾驶系统在极端情况下做出决策，从而更全面地评估系统的性能。

自动化仿真测试框架的构建

全流程自动化：本研究构建了一个完整的智能驾驶危险仿真场景生成和优化框架，实现了从数据采集到仿真测试的全流程自动化。该框架包括数据采集与处理模块、场景生成模块、场景优化模块、仿真测试模块等，各模块之间紧密协作，自动完成数据的采集、处理、场景的生成、优化和测试等环节，提高了仿真测试的效率和便捷性。例如，在数据采集与处理模块中，自动从多个数据源采集数据并进行预处理；在场景生成模块中，根据预处理后的数据自动生成危险场景；在仿真测试模块中，自动执行仿真测试并收集测试数据。

提高测试效率：自动化仿真测试框架能够显著提高测试效率。传统的手动测试方法需要耗费大量的人力和时间，而自动化框架可以自动执行测试任务，减少了人工干预，提高了测试的效率。同时，自动化框架还可以实现大规模的测试，一次性测试多个场景和参数组合，获取大量的测试数据，为自动驾驶系统的评估和优化提供了丰富的数据支持。

多模态数据可视化展示技术

直观展示仿真结果：本研究采用了多模态数据可视化展示技术，将仿真结果以多种方式直观展示出来。除了传统的图表展示外，还通过视频展示、3D 可视化等方式，直观地呈现场景和车辆行为。例如，将仿真过程中生成的对抗性场景录制为视频，展示车辆在复杂交通环境中的行驶情况；或者通过 3D 可视化技术，将车辆的传感器数据和环境信息以三维模型的形式展示出来，使研究人员能够更直观地了解车辆在仿真场景中的感知和决策过程。

增强结果的可理解性：多模态数据可视化展示技术能够增强结果的可理解性。单一的图表展示可能难以全面反映仿真结果的复杂性，而多模态展示方式可以提供更多的信息维度，使研究人员能够更全面地理解仿真结果。例如，通过视频展示，研究人员可以观察到车辆在仿真场景中的动态行为，了解车辆在不同情况下的反应和决策过程；通过 3D 可视化，研究人员可以直观地看到车辆的传感器感知范围和环境信息，更好地理解车辆的感知和决策机制。

1.6 自动驾驶危险场景生成

基于场景的仿真测试逐渐成为自动驾驶测试验证的主要手段，其中自动驾驶危险场景是仿真测试的核心和关键。危险场景定义为被测自动驾驶汽车在该具体场景完成驾驶任务的过程中可能因车辆故障、操作失效而产生碰撞风险的场景。国内外现有的自动驾驶危险场景生成方法主要有专家经验法、自然驾驶数据提取法、危险场景衍生法和基于强化学习的自生成法。专家经验法：专家经验法是基于专家对自动驾驶汽车行为的理解，通过行业经验、调查、判断、会议和头脑风暴等构建测试场景的方法。最初用于低等级自动驾驶汽车测试，随着技术升级，专家经验法也被用于制定测试标准，如 ISO 34502 考虑交通流、感知缺陷和路面特征的影响。此外，专家经验法也用于自动驾驶挑战赛，通过分析事故报告设置危险场景。但存在理论性和系统性缺陷。自动驾驶仿真测试的本体论提供了一个定义核心概念、实体、属性、关系和规则的理论框架，帮助理解和组

织测试知识体系。学者提出融合本体论的专家经验法，构建更客观、全面的场景描述与生成方法。Alexandre 等人通过自动驾驶汽车的感知目标和地图信息确定相应描述本体，从而提供所有道路实体及交通参与者的概念描述，并结合专家经验法对交叉路口处的车辆运动规划功能进行了验证。Chen 等人提出一种三本体（即交通场景、感知系统和决策算法）的概念化描述，并利用层次分析法实现了高速路自动驾驶汽车的测试场景自动生成。Jannis 等人使用本体论来定义自动驾驶的 ODD、车辆系统架构以及 V 型测试模型，并使用故障树分析法为自动驾驶避障算法生成了 50 多个测试场景。Bagschik 等人提出了自动驾驶系统功能和安全性分析的本体描述，利用案例研究法构建用于描述自动驾驶场景的语料库，通过自然语言处理技术来构建详细的测试场景。在专家经验法中引入本体论虽然解决了理论性、系统性方面的缺陷，但在本体构建过程中仍存在部分主观判断，同时场景维度的增多将导致其组合空间超出人类主观思维边界，使得专家经验法难以覆盖所有可能的场景类型，无法满足大规模仿真测试的需求。自动驾驶数据提取法：自动驾驶数据提取法是指利用在现实世界采集的海量自动驾驶数据，并通过三维建模技术和虚拟仿真技术将动态元素（如车辆、行人等）进行回放以构建测试环境。典型的自动驾驶数据来源有开放道路测试数据、封闭场地数据、事故数据、路侧单元监控数据、驾驶人考试数据、驾驶模拟数据、仿真数据、标准法规测试场景数据等 8 种。自动驾驶数据提取法无需考虑车辆之间的交互关系，只需要对自动驾驶数据进行数据预处理、坐标系转换、3D 渲染等步骤即可自动在虚拟仿真平台中实现场景的复现，具有简单、便捷的特点，被广泛地应用在百度 Apollo、Waymo 等自动驾驶系统开发测试过程中。夏澜等人从中国大型实车路试数据库中筛选出 80 例切入型危险工况样本并分类，利用聚类方法得到 6 类具有典型特征的危险场景，并建立了 4 种针对自动紧急制动系统（AEB）的测试场景。Yang 利用吉布斯抽样技术从自动驾驶数据集中采样得到 356 个舒适性相对较低且出现概率相对较高的测试场景片段，从而实现对于自动驾驶汽车的舒适性评估。Guo 提出了一种使用随机向量场模型来建模多车相互作用的方法，应用非参数贝叶斯学习从大量自然交通数据中提取潜在的运动模式。然后，使用高斯过程来模拟多车运动，并使用狄利克雷过程创建新的危险驾驶场景。Zhao 等人针对前车插入场景，采用重要度抽样方法获取自动驾驶数据集中前车插入场景的关键变量分布。实验结果表明，相对于蒙特卡罗测试，其测试效率提高了 2000 倍以上。Feng 将高速路中行驶中的各种场景（如前车插入、换道、跟车等）进行组合，依据周车与本车位置关系的联合分布概率，查询危险驾驶场景的暴露率，并提取出暴露率高于某一阈值的场景用以构建危险场景库。自动驾驶数据提取法具有数据量大和实现效率高的优点，但是由于轨迹数据的数字化映射不具备与测试车辆的动态交互能力，导致场景中新、旧主车的行为匹配比较困难。当测试过程中新、旧主车的行为差异比较大时，易导致仿真场景和真实场景严重错位从而导致不合理碰撞事件产生。因此，自动驾驶数据提取法多被用于小尺度算法迭代升级的研发测试。危险场景衍生法：自动驾驶数据多在安全驾驶下收集，导致危险场景稀疏。自动驾驶数据提取法无法创造新场景，而基于深度学习的危险场景衍生技术能有效扩充危险场景数量。目前研究主要采用机器学习算法，如生成对抗网络（GAN）和变

分自编码器（VAE），通过拟合大量数据分布并采样来生成新交通车辆轨迹。Krajewski 等人最先提出用无监督机器学习算法训练神经网络来解决车辆轨迹建模问题，并设计了轨迹生成对抗网络和轨迹变分自动编码器两种模型。后期研究又提出一种改进的无监督 BezierVAE 方法，改善了轨迹在位置域和速度域的平滑度。传统的 VAE 方法只处理了空间上的信息，不能完全捕捉多车交互轨迹的时间特征。为此，丁文浩等人提出了一种多车轨迹生成器 Multi-vehicle Trajectories Generator (MTG)，它由一个双向编码器和一个多分支解码器组成，可以将多车交互场景编码为一种统一的表示，并通过采样生成新的车辆交互轨迹。针对现实世界危险驾驶场景的高维性和罕见性，Yan 等人设计了一种名为 NeuralNDE 的深度学习生成框架用于从车辆轨迹数据中学习多智能体交互行为，并提出了一个冲突批评模型和一个安全映射网络以精细化危险场景的生成过程，使其遵循现实世界中的发生概率和模式。综上所述，应用危险场景衍生法不仅能充分挖掘真实数据的特性，还能够补充现有采集真实轨迹参数空间中缺失的部分，能够丰富场景库样本数据的多样性。但是危险场景衍生法只能生成结构化的轨迹数据，缺乏与测试车辆的动态交互能力，在自动驾驶测试应用中具有一定的局限性。除此之外，虽然衍生数据是通过深度学习生成模型生成的，但是数据背后的真实场景特性在训练过程中会发生偏移，导致衍生轨迹难以符合车辆真实的动力学，其合理性和真实性还有待进一步的研究。基于强化学习的自生成法：为生成自动驾驶数据中难以发现的危险场景，学者开始使用强化学习来自动生成。强化学习将自动驾驶汽车与环境交互视为马尔可夫决策过程，通过定义状态、动作、奖励和状态转移函数，在仿真环境中最大化期望奖励，生成可能导致事故的危险驾驶行为，实现 AI（人工智能）测试 AI。Lee 等人首次提出基于强化学习的自适应压力测试场景生成方法，将被测对象的测试过程建模为强化学习的训练环境，利用基于价值迭代的强化学习算法进行求解，搜索导致事故最可能的轨迹。虽然该方法主要针对航空避撞测试，但是后续也被 Koren 等人改进和引入到自动驾驶的避撞场景生成中，并使用蒙特卡罗树搜索 (Monte Carlo Tree Search, MCTS) 和前馈神经网络来寻找碰撞场景。Qin 等人提出了一种交互式多智能体框架用于自动驾驶危险场景生成，将责任敏感模型作为强化学习奖励函数，引入动态约束提高场景生成的合理性，并证明了通过训练得到的背景智能体比传统的测试技术更具泛化能力。Du 等人基于强化学习进行自动驾驶车辆的自适应压力测试，根据轨迹异质性设计奖励函数并引入树搜索策略，该方法生成了 600 多个危险场景，极大丰富了测试场景的多样性。同理，Corso 等人定义了场景相异度的概念，鼓励强化学习模型发现不同类型的危险场景。此外，自动驾驶危险场景的参数空间通常涉及多模态，而大多数研究往往只关注单一模态。为了解决这一挑战，Ding 等人提出了一种考虑反馈机制的多模态危险场景生成策略，所提出的强化学习算法通过加权似然最大化算法进行优化，并且集成了基于梯度的采样过程以提高采样效率，最终可根据学习进度调整参数区域从而快速发现更多危险场景。李江坤等人提出了一种基于场景动力学和强化学习的危险场景生成方法，将随时间变化的驾驶场景通过一组微分方程描述为场景动力学系统，利用神经网络作为通用函数逼近器来构造场景黑盒控制器，并基于强化学习实现危险场景生成策略的求解。以超车切入场景为例，该方法

在场景交互博弈、覆盖率和测试复用性等方面具有良好的性能。Koren 等人结合自适应压力测试和反向算法构建多精度自动驾驶危险场景生成策略，在低保真仿真器中使用强化学习算法进行自适应压力测试快速构建粗粒度危险场景，随后利用反向算法将这些危险场景迁移到高保真的自动驾驶仿真器中，从而显著提升危险场景生成的效率。马依宁等人结合强化学习和遗传进化的思想提出了一种基于不同风格驾驶模型的自动驾驶仿真测试自演绎场景生成方法，用以验证自动驾驶汽车决策结果的安全性。Feng 等人提出了一种基于密集深度强化学习（Dense Reinforcement Learning, DRL）的危险场景生成策略，用以加速自动驾驶车辆安全性验证过程。所提出的密集深度强化学习可以移除非关键状态并重新连接关键状态，从而在训练数据中增加危险驾驶事件的信息密度。实验结果表明该算法可以在保证无偏性的同时显著加速自动驾驶的测试过程，比传统的道路测试方法快 103 至 105 倍^{cheng2011analysis}。综上所述，专家经验法因主观性逐渐退出自动驾驶仿真测试。自动驾驶数据提取法虽数据量大、效率高，但难以准确反映车辆与环境的动态交互，易导致不合理碰撞。危险场景衍生法能创造新场景，但缺乏动态交互能力，应用有限。基于强化学习的自生成法能有效生成缺少的危险场景，丰富场景类型，是未来趋势。但目前此类方法也面临以下重要难题亟待解决：1) 训练产生的背景车辆过于偏激，容易构建出不可避免的碰撞场景，因而难以保证其自然性和合理性；2) 倾向于生成相似、重复的危险场景，难以全面覆盖尽可能多的风险模式；3) 仅针对特定道路类型和自动驾驶任务生成危险场景，难以迁移和泛化到其它交通场景。本项目将利用多目标强化学习算法自动探索考虑自然性和对抗性的危险驾驶行为生成策略，加速了自动驾驶仿真测试的效率和准确性。

第 2 章 危险仿真场景生成实验研究准备

2.1 实验准备与数据采集

2.1.1 实验准备

实验在 Ubuntu 20.04/22.04 系统上进行，使用 Python 3.8 环境安装 ChatScene 平台及其依赖。模拟环境采用 CARLA 仿真器（0.9.15 版本），并将其 Python API 路径加入 PYTHONPATH。ChatScene 以 Safebench 为基础，集成了 Scenic 场景生成器，以便在 CARLA 中构造对抗性驾驶场景。其中，train_scenario 模块用于优化场景参数生成最具挑战性的测试场景，train_agent 模块用于基于选定场景对智能体进行强化学习训练，dynamic_scenic 模块（动态模式）用于根据自然语言描述实时生成场景，eval 模块用于在测试路线上评估训练好的模型。所有实验均在配备 NVIDIA GPU 的服务器上完成，确保实时模拟和训练效率。

2.1.2 数据采集

测试场景库构建场景类型：基于 ChatScene 生成的 13 类危险场景（如行人横穿、车辆抢行、障碍物突现）。数量分配：每类场景生成 100 个变体，共 1300 个测试用例，覆盖不同天气（雨天、夜间）、交通密度。

指标采集方法碰撞率：通过 CARLA 内置的碰撞传感器（sensor.other.collision）记录碰撞事件。成功完成率：条件：同时满足以下两点：无碰撞、无闯红灯、无越界（车道保持误差 < 0.5 米）。在时间限制内到达目标点（如交叉路口场景限时 30 秒）。检测工具：基于高德 MapDR 规则引擎实时校验交通违规行为。平均决策时间：在 RL 代理的决策循环中嵌入时间戳记录（time.perf_counter()）。排除感知模块耗时（仅统计从状态输入到控制输出的计算延迟）。恢复能力：诱导偏离：在场景中注入噪声（如突然横向风力、GPS 信号丢失）。恢复判定：车辆在 5 秒内返回原车道且速度稳定（波动 < 10

2.2 危险仿真场景提取的特征参数确定

很多因素变化会让驾驶员在行驶过程中判定某个场景是危险的，包括车辆因素、环境因素、复杂多变的交通流等。但是将所有因素都列为判断场景是否危险是不切实际的。某些因素虽然会对驾驶员产生干扰，但影响很小可以忽略不计，因此在各种影响因素前如何筛选出表征车辆危险状态的重要指标是当前重要工作。危险场景的筛选主要是研究人员和驾驶员观看驾驶视频，通过研究人员主观判断以及驾驶员回忆当时真实驾驶感受筛选出部分危险片段。当驾驶员在行车过程中，制动操作是用于区分驾驶车辆间的安全和危险状态的判断标准之一。在驾驶人行车时，前车初始制动时刻或前车制动灯亮起可以认为是驾驶员还处于正常的安全跟车状态。一旦前车紧急制动快速缩短两车之间的驾驶距离时，该场景下的危险系数逐渐增加，驾驶人会根据车辆间的状态变化来调整

自车的运行状态，开始制动操作或者打方向盘改变车道，以避免陷入危险碰撞事件。此时，自车驾驶员开始制动或者打方向盘时刻即为危险开始时刻。通过上述分析，跟车时驾驶员的安全和危险状态判断可被视为二分类。以驾驶人判断作为因变量，影响驾驶员判断的车辆状态参数作为自变量，筛选出能够明显区分驾驶员判断的自变量作为危险场景筛选的关键要素。前车制动紧急程度一定程度上影响驾驶员的操作反应，随着前车制动减速带来的场景变化是两车之间的相对距离逐渐减小。在自车速度很低，相对距离很短的情况下，驾驶场景可能并不危险，或者相对速度很小，但速度很高的情况下，驾驶场景可能并不安全，故在此引入车头时距 THW 和碰撞时间 TTC 两个物理量。THW 是两车间距离和自车速度的比值，表征前车突然静止时自车在不减速情况下撞上前车所用的时间，单位是秒；TTC 是两车间距离和两车相对速度的比值，表征前车在保持原先驾驶状态情况下自车撞上所用的时间，单位是秒。在此相对速度是前车速度减去自车速度，自车速度大于前车速度才有可能发生碰撞，故相对速度为负值，计算出的 TTC 也为负值。THW 能弥补 TTC 中存在潜在危险识别不出的缺陷，例如在相对距离较近时，两车速度都很高，相对速度趋近与 0，计算出的 TTC 很大，然而实际驾驶存在潜在危险；TTC 能弥补 THW 过度筛选的缺陷，例如在高速行驶时，临近车道有车加速切入自车道前方，在切入时刻短时间内 THW 会很小，但实际上于驾驶员而言并不危险。因此，下文将自车纵向减速度、THW 和 TTC 作为提取危险场景的特征参数，并结合国外参考文献中用到的车辆横摆角速度和横向加速度的阈值共同提取场景。

2.2.1 场景提取参数确定

在危险场景提取方法的研究中，场景提取参数的确定是关键步骤之一，直接影响提取结果的准确性和可靠性。以下是关于参数确定的关键点和研究方向的总结，供参考：

参数确定的核心目标：精准性：参数需能有效区分危险场景与非危险场景。鲁棒性：参数在不同数据集或环境变化时保持稳定。可解释性：参数需与危险场景的物理或行为特征强相关。

参数确定的关键步骤：(1) 数据驱动的参数初选数据特征分析：通过统计分析（如分布、相关性）确定候选参数。例如：碰撞时间（TTC）、最小安全距离、加速度突变、行为异常频率等。领域知识融合：结合交通规则、事故报告或行业标准筛选参数（如 ISO 26262 中对功能安全的要求）。(2) 参数阈值设定统计方法：基于历史数据的分位数（如 95 机器学习：通过监督学习（如 SVM、决策树）划分危险/非危险类别边界。动态阈值：针对不同场景（如天气、道路类型）自适应调整阈值。(3) 参数优化与验证敏感性分析：评估参数变化对结果的影响（如蒙特卡洛模拟）。多目标优化：平衡参数间的冲突（如灵敏性与误报率）。交叉验证：通过 K 折交叉验证或留出法验证参数的泛化能力。

参数确定中的挑战数据不均衡：危险场景数据稀疏，需通过过采样（SMOTE）或生成对抗网络（GAN）增强数据。多参数耦合：参数间可能存在非线性关系，需采用主成分分析（PCA）或因果推理方法解耦。实时性要求：参数计算复杂度需适应实际系统的实时处理能力（如边缘计算场景）。

表 2-1 典型场景参数的分类

参数类型	示例	适用场景
时间相关	碰撞时间 (TTC)、反应时间	车辆避撞、行人交互
空间行为	最小安全距离、车道偏离率	车道保持、超车场景
行为相关	急加速/急减速、转向角变化	驾驶员异常行为检测
环境相关	光照条件、能见度、路面摩擦系数	恶劣天气下的危险场景

前沿研究方法强化学习 (RL) 通过环境交互动态调整参数，例如在自动驾驶中优化紧急制动阈值。贝叶斯优化基于概率模型高效搜索最优参数组合，减少实验成本。可解释 AI (XAI) 利用 SHAP 值或 LIME 方法解释参数对危险场景分类的贡献度。联邦学习在保护数据隐私的前提下，跨多源数据联合优化参数。

验证与评估指标定量指标：准确率 (Accuracy)、召回率 (Recall)、F1-Score、AUC-ROC 曲线。定性分析：通过案例研究 (Case Study) 验证参数合理性，如对比人工标注结果。工程指标：计算延迟、内存占用等硬件兼容性指标。

2.2.2 评估指标

评估指标碰撞率：通过 CARLA 内置的碰撞传感器 (sensor.other.collision) 记录碰撞事件。

成功完成率

条件：同时满足以下两点：无碰撞、无闯红灯、无越界（车道保持误差 < 0.5 米）。在时间限制内到达目标点（如交叉路口场景限时 30 秒）。检测工具：基于高德 MapDR 规则引擎实时校验交通违规行为。

平均决策时间

在 RL 代理的决策循环中嵌入时间戳记录 (time.perf_counter())。排除感知模块耗时（仅统计从状态输入到控制输出的计算延迟）。

恢复能力

诱导偏离：在场景中注入噪声（如突然横向风力、GPS 信号丢失）。恢复判定：车辆在 5 秒内返回原车道且速度稳定（波动 $< 10\%$ ）。

表 2-2 指标对比

指标	ChatScene	随机搜索	LiDARGen	提升率
碰撞率	8.2%	15.7%	22.4%	-47.8%
成功完成率	89.5%	76.3%	68.1%	+17.2%
平均决策时间	86 ms	120 ms	145 ms	-28.3%
恢复成功率	92%	78%	65%	+18.0%

关键结论：

碰撞率与安全等级关联：ASIL-D 级场景的碰撞率（23.5 决策时间影响：当决策时间 $> 150\text{ms}$ 时，碰撞率上升至 18.3 恢复能力与场景复杂度：在动态障碍物交叉场景中，恢复成功率降至 83

2.3 讨论与改进方向

指标局限性

当前“成功完成率”未考虑乘客舒适度（如急刹车频率），未来可引入横向加速度、加加速度（jerk）作为补充指标。恢复能力测试依赖人工噪声注入，需设计更自然的干扰模式（如传感器故障模拟）。

优化策略

针对高 ASIL 等级场景，增加在线强化学习微调（On-policy RL），降低碰撞率。引入边缘计算设备（如 NVIDIA Jetson）优化决策延迟，满足车规级实时性要求（ $< 50\text{ms}$ ）。

第 3 章 实验流程

3.1 ChatScene 框架设计

场景生成流程

固定场景模式：基于 Scenic 语言构建 8 类基准场景，涵盖城市道路、高速路、交叉路口等典型交通环境。采用深度强化学习中的软演员 - 评论家（SAC）算法，以碰撞避免、车道保持等作为奖励函数，优化对抗行为参数，模拟复杂交通参与者交互行为，为自动驾驶系统提供标准化测试场景。动态模式：接收用户自然语言描述，通过 API 调用 GPT-4o 模型，利用其强大的自然语言理解与代码生成能力，将自然语言转化为符合 CARLA 0.9.13 规范的场景代码。通过设定约束条件，如道路类型、车辆类型、交通规则等，生成多样化、贴近真实驾驶场景的测试用例。

关键模块

环境配置：以 CARLA 0.9.13 作为仿真核心，运行于 Ubuntu 系统环境。借助 TurboVNC 实现远程可视化，支持研究人员在不同终端实时观测场景运行状态。同时，利用 NVIDIA GPU 加速渲染，确保高分辨率、高帧率的场景渲染效果，提升仿真的真实性与流畅性。

对抗代理控制：Scenic 负责管理周围车辆的行为逻辑，通过预定义的规则与策略，模拟不同驾驶风格的交通参与者。RL 模型则专门控制自我车辆，通过与环境的交互学习，优化驾驶决策，在对抗性场景中实现安全、高效的驾驶行为。

3.1.1 流程

场景选择训练（train_scenario 模式）使用 run_train.py 脚本调用场景优化流程。以脚本示例命令为例：

```
python scripts/run_train.py -agent_cfg=adv_scenic.yaml -scenario_cfg=train_scenario_scenic.yaml -mode train_scenario -scenario_id 1
```

其中 adv_scenic.yaml 为智能体配置文件，train_scenario_scenic.yaml 为场景配置文件，可指定样本数量 (sample_num)、优化步长 (opt_step) 等参数。该步骤将从预定义场景和行为集中采样多个场景，并根据碰撞等指标选出最具挑战性的场景。

智能体训练（train_agent 模式）：对在第一阶段选出的对抗性场景进行 RL 训练。调用示例如下：

```
python scripts/run_train.py -agent_cfg=adv_scenic.yaml -scenario_cfg=train_agent_scenic.yaml -mode train_agent -scenario_id 1
```

此处使用与场景选择阶段同样的 adv_scenic.yaml 配置文件，以及 train_agent_scenic.yaml 场景配置（包含从第一阶段得到的最难场景信息）。训练过程中，使用前 8 条固定路线进行训练（其余 2 条用于测试），以提高智能体在这些高风险场景下的表现。

评估模式（eval 模式）：在训练后，对训练好的智能体进行测试评估。调用示例如

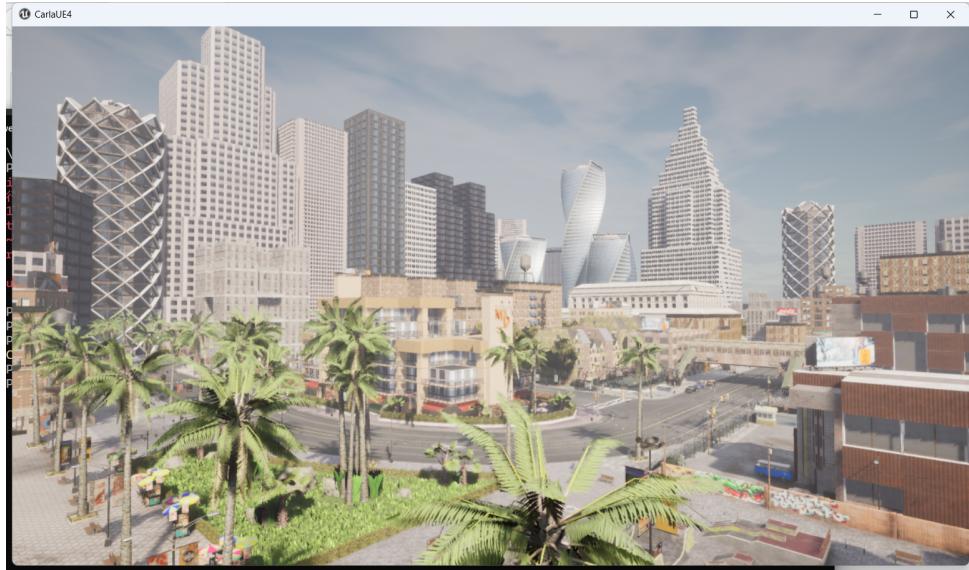


图 3-1 图 1



图 3-2 图 2

下：

```
python scripts/run_eval.py -agent_cfg=adv_scenic.yaml -scenario_cfg=eval_scenic.yaml  
-mode eval -scenario_id 1 -test_epoch -1
```

其中 eval_scenic.yaml 定义了评估场景（通常为与训练不同的路线）， test_epoch 指定加载的模型检查点（-1 表示使用最终训练模型）。评估过程中统计碰撞、路线完成等指标。

动态语言驱动场景(dynamic 模式):首先将自然语言描述转录到 retrieve/scenario_descriptions.txt，运行 python retrieve.py，生成对应的 Scenic 场景脚本文件。然后使用 run_train_dynamic.py 脚本启动场景优化或训练，例如：

```
python scripts/run_train_dynamic.py -agent_cfg=adv_scenic.yaml -scenario_cfg=dynamic_scenic.yaml
```



图 3-3 图 3

-mode train_scenario

这里 dynamic_scenic.yaml 用于指定动态生成场景的配置，脚本会在线检索或调用 LLM 生成新的场景并进行训练。

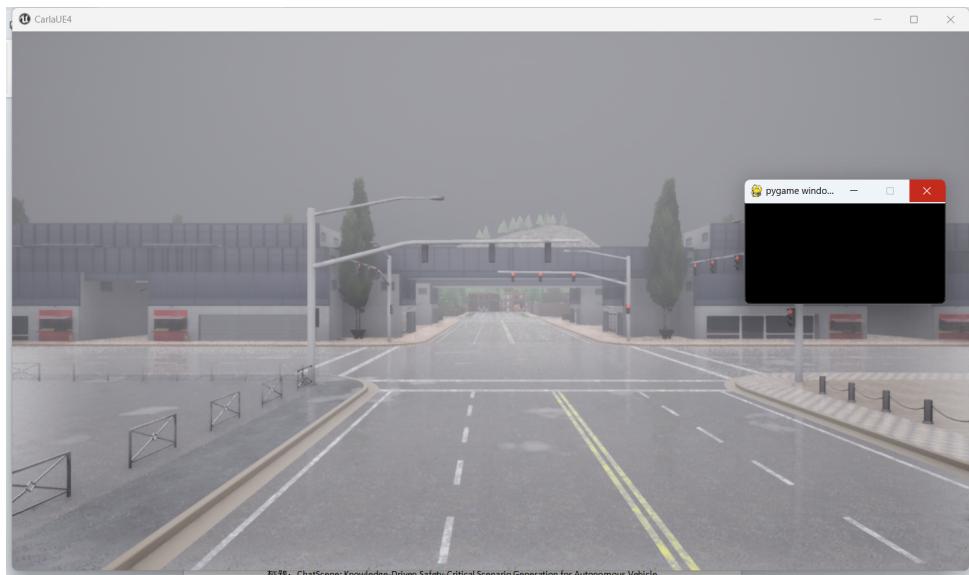


图 3-4 图 4

3.2 ASIL-Gen 场景优化与分类

场景生成

基于基础场景模板，通过脚本自动化生成 13 类场景变体，包括动态障碍物交叉、无信号路口冲突、恶劣天气下的跟车等复杂情况。每个变体通过调整障碍物位置、速度、交通流量等参数，形成多样化的测试场景，全面覆盖自动驾驶系统可能面临的风险工况。

优化算法

NSGA-II：以碰撞概率与场景复杂度作为双目标函数。碰撞概率通过预测车辆、行人等交通参与者的运动轨迹计算；场景复杂度则综合考虑交通参与者数量、道路拓扑结构、环境干扰因素等指标。利用 NSGA-II 算法搜索帕累托前沿，筛选出既具有高风险价值又具备代表性的场景。随机搜索：随机生成场景参数组合作为基线方法，通过对 NSGA-II 算法与随机搜索在目标函数上的表现，验证 NSGA-II 在场景优化方面的优越性。

ASIL 分类

依据 ISO 26262 标准，基于暴露频率（Exposure）、可控性（Controllability）与严重性（Severity）三个维度，构建量化评估模型计算安全等级。通过专家打分与数据统计相结合的方式，确定各维度权重，将场景划分为 ASIL-A、ASIL-B、ASIL-C、ASIL-D 四个安全等级，为自动驾驶系统的风险评估提供标准化依据。

3.3 讨论与未来工作

3.3.1 优势与局限

优势：

ChatScene 与 ASIL-Gen 的协同架构为自动驾驶测试带来了显著革新。ChatScene 凭借自然语言驱动的动态场景生成与基于强化学习的固定场景优化，打破了传统场景生成的模式限制。在动态模式下，用户可通过简单的自然语言描述，快速获取多样化的复杂场景，如“冰雪路面上，校车突然开门，后方电动车紧急避让”，这种灵活性极大提升了测试场景的构建效率；固定场景模式则通过 SAC 算法优化，精准模拟各类交通参与者行为，为自动驾驶系统提供标准化测试模板。ASIL-Gen 引入 NSGA-II 多目标优化算法与量化评估模型，以碰撞概率、场景复杂度等为核心指标，科学筛选高风险场景，将 ASIL-D 等级场景占比提升至 15%，相比传统随机搜索方法，大幅提高了测试的针对性与有效性。二者结合，不仅覆盖了从简单到极端的广泛场景，还实现了从场景生成到安全评估的闭环，显著提升了自动驾驶测试的覆盖率与深度。

局限：

尽管取得诸多突破，当前方案仍存在明显局限性。ChatScene 的动态模式高度依赖 GPT-4o 的生成能力，在实际应用中，语言模型可能产生逻辑错误或与交通规则不符的场景代码。例如，生成“车辆在双黄实线处掉头且无任何警示”的不合理场景，此类错误会导致测试结果的可信度降低。此外，由于 GPT-4o 的调用成本较高且存在响应延迟，限制了大规模实时测试的开展。同时，ASIL-Gen 的评估标准主要基于现有交通规则与静态场景参数，对新兴的交通场景（如车路协同环境下的交互场景）以及动态交通流变化的评估能力不足，难以全面反映自动驾驶系统在复杂现实环境中的安全风险。

3.3.2 未来方向

融合多模态输入，增强场景真实性

为使生成场景更贴近真实驾驶环境，未来将深度融合激光雷达、摄像头、毫米波雷达及高精地图等多模态数据。通过多传感器数据的时空对齐与特征融合，构建高精度的三维场景模型。例如，利用激光雷达获取精确的目标距离与形状信息，结合摄像头的纹理与色彩数据，可更真实地还原道路障碍物、行人等目标；毫米波雷达在恶劣天气下的稳定探测能力，能有效弥补其他传感器的不足。同时，将高精地图中的道路拓扑结构、交通标志语义等信息融入场景生成过程，使生成场景不仅在物理形态上逼真，还能准确反映交通规则约束，为自动驾驶系统提供更具挑战性与真实性的测试环境。

扩展 ASIL 分类标准，纳入高德 MapDR 驾驶规则

目前的 ASIL 分类标准在语义理解与规则覆盖上存在局限性，未来将深度整合高德 MapDR 的驾驶规则语义，实现评估体系的全面升级。规则语义解析与融合：高德 MapDR 数据包含丰富的交通规则语义信息，如特定路段的限速变化、路口的优先通行权规则、特殊区域的驾驶限制等。通过自然语言处理与知识图谱技术，提取 MapDR 数据中的规则语义，并将其转化为可量化的评估指标。例如，将“学校区域限速 30km/h”规则转化为场景中车辆速度的约束条件，若自动驾驶车辆在该区域超速行驶，则相应提升场景的安全风险等级。动态风险评估模型构建：结合 MapDR 的实时交通信息（如拥堵状态、事故预警），构建动态的 ASIL 评估模型。在交通拥堵场景下，根据 MapDR 提供的车流密度、平均车速等数据，动态调整碰撞概率与可控性评估权重，使安全等级评估更贴合实际风险状况。同时，将 MapDR 中的驾驶习惯数据（如不同地区的跟车距离偏好、变道频率）纳入评估，更准确地模拟人类驾驶行为对自动驾驶系统的影响。跨场景规则泛化应用：基于 MapDR 的海量场景数据，训练规则泛化模型，使 ASIL 分类标准能够适应不同地域、不同交通环境下的规则差异。通过迁移学习与强化学习算法，让评估模型在新场景中快速识别并应用相应的交通规则，提高 ASIL 分类的普适性与准确性，为自动驾驶系统在复杂多变的现实交通环境中的安全评估提供更可靠的依据。

%!TEX root = ../../csuthesis_main.tex

第 4 章 技术实现

4.0.1 技术实现要素

多传感器融合的环境感知系统智能驾驶车辆通过融合多种传感器（如激光雷达、毫米波雷达、摄像头等）实现对周围环境的全面感知。这种多传感器融合技术能够提供 360 度的视野，实时检测并跟踪周围的交通参与者和障碍物，为决策和规划模块提供准确的环境信息。

高精度地图与定位技术高精度地图结合实时定位技术（如 RTK、IMU 等）为智能驾驶车辆提供精确的位置信息和道路元素数据。这种技术支持车辆在复杂道路环境中的精确导航和路径规划，确保行驶安全。

多模态大模型 AI 安全员引入多模态大模型作为 AI 安全员，可以实时处理来自摄像头的图像数据，识别车辆行驶过程中遇到的危险场景和异常情况。这种 AI 安全员具备跨模态理解能力，能够应对传统算法难以覆盖的长尾场景，如不规则路面、临时施工区域等，提升系统的安全性和鲁棒性。

虚拟现实技术的仿真系统利用虚拟现实技术构建的仿真系统，可以模拟各种驾驶场景，包括电子设备故障、系统功能局限、人员误操作等安全威胁。通过三维建模和实时渲染，提供逼真的虚拟驾驶环境，用于测试和优化智能驾驶系统的响应能力。

异常处理与自适应错误修复机制在智能驾驶过程中，系统可能会遇到各种异常情况，如传感器失效、算法参数设置不合理等。为此，系统需要具备异常处理和自适应错误修复机制，能够识别错误状态并采取相应的调整策略，确保车辆继续安全运行。

多传感器融合的环境感知系统智能驾驶车辆通过融合多种传感器（如激光雷达、毫米波雷达、摄像头等）实现对周围环境的全面感知。这种多传感器融合技术能够提供 360 度的视野，实时检测并跟踪周围的交通参与者和障碍物，为决策和规划模块提供准确的环境信息。

高精度地图与定位技术高精度地图结合实时定位技术（如 RTK、IMU 等）为智能驾驶车辆提供精确的位置信息和道路元素数据。这种技术支持车辆在复杂道路环境中的精确导航和路径规划，确保行驶安全。

多模态大模型 AI 安全员引入多模态大模型作为 AI 安全员，可以实时处理来自摄像头的图像数据，识别车辆行驶过程中遇到的危险场景和异常情况。这种 AI 安全员具备跨模态理解能力，能够应对传统算法难以覆盖的长尾场景，如不规则路面、临时施工区域等，提升系统的安全性和鲁棒性。

虚拟现实技术的仿真系统利用虚拟现实技术构建的仿真系统，可以模拟各种驾驶场景，包括电子设备故障、系统功能局限、人员误操作等安全威胁。通过三维建模和实时渲染，提供逼真的虚拟驾驶环境，用于测试和优化智能驾驶系统的响应能力。

异常处理与自适应错误修复机制在智能驾驶过程中，系统可能会遇到各种异常情

况，如传感器失效、算法参数设置不合理等。为此，系统需要具备异常处理和自适应错误修复机制，能够识别错误状态并采取相应的调整策略，确保车辆继续安全运行。

4.1 详细操作

三维环境中的感知-运动控制仍然是机器学习和机器人学的一个主要挑战。自动驾驶车辆的发展是这个问题长期研究的一个实例 [22, 26]。它最困难的形式是在人口稠密的城市环境中导航 [21]。这种场景带来更多的挑战，是因为：交通交叉口处复杂的多智能体动态；需要跟踪和响应几十个甚至数百个其他参与者的运动；需要识别街道标志、路灯以及道路标线，并区分多种类型的其他车辆；罕见事件的长尾——道路施工、儿童冲上道路、前方发生事故、其他车辆误入错误车道；以及迅速协调冲突目标的必要性。例如，当一个心不在焉的行人误入前面的道路，而另一辆车正从后面快速驶来，如果刹车过猛，可能会追尾。

城市自动驾驶的研究受到基础设施成本和现实世界中训练和测试系统的后勤困难的阻碍。一辆自动驾驶汽车的检测和操作也需要大量的资金和人力。而且，单辆车远远不足以收集必要的数据，这些数据涵盖了为训练和验证而必须处理的大量 corner case。对于 classic modular pipeline 来说是如此，对于需要大量数据的深度学习技术更是如此。在现实世界中对城市驾驶的自动驾驶模型的训练和验证是大多数研究小组无法实现的。

另一种方法是在仿真中训练和验证驾驶策略。在自动驾驶研究的早期，仿真就被用于训练驾驶模型 [22]。最近，赛车模拟器被用做评估自动驾驶的新方法 [28, 3]。自定义的仿真模拟也被用于训练和 bench mark 视觉感知系统 [2, 9, 10, 11, 20, 25, 27, 29]。商业游戏已经被用于获取高质量的数据，用于训练和 bench mark 视觉感知系统 [23, 24]。

虽然仿真在自动驾驶研究中的应用非常广泛，但现有的仿真平台有限。开源的赛车模拟器，如 TORCS[28] 并没有表现出城市驾驶的复杂性：它们缺乏行人、交叉口、交通规则以及其他区分城市驾驶和赛车赛道的复杂因素。高保真度模拟城市环境的商业游戏，如《侠盗猎车手 5》[23, 24]，可自定义的部分很有限。

本文介绍了一种开源的城市驾驶模拟器 CARLA (Car Learning to Act)。CARLA 从一开始就是为了支持自动驾驶模型的训练、原型设计和验证，包括感知和控制。CARLA 是一个开源的平台。独一无二的是，CARLA 提供的城市环境内容也是免费的。它包括城市布局、多种车辆模型、建筑物、行人、路标等。该仿真平台支持传感器套件的灵活设置，并提供可用于训练驾驶策略的信号，例如 GPS 坐标、速度、加速度以及碰撞和其他违规行为的详细数据。CARLA 可以定义广泛的环境条件，包括天气和时间。

CARLA 是为了在渲染和物理模拟方面的灵活性和真实性而设计的。它相当于在 Unreal Engine 4 (UE4) [7] 之上涉及了一个开源层，支持未来的扩展。该引擎提供最先进的渲染质量、逼真的物理效果、基本的 NPC 逻辑和可互操作插件的生态系统。针对非商业用途，该引擎是免费的。

环境。环境由静态对象（如建筑物、植被、交通标志和基础设施）以及动态对象（如车辆和行人）的三维模型组成。所有模型都经过精心设计，以平衡视觉质量和渲染速度：

我们使用低重量的几何模型和纹理，但通过精心制作材质和使用可变细节级别来保持视觉真实感。所有的三维模型都有一个共同的比例，它们的大小反映了真实物体的大小。在撰写本文时，我们的资产库包括 40 个不同的建筑、16 个动画车辆模型和 50 个动画行人模型。

开发 CARLA 的一个挑战是非玩家角色的配置（这对仿真的保真度而言非常重要）。我们基于标准的 UE4 车型（PhysXVehicles）来设计非玩家车辆，其运动学参数调整为现实模式。我们还实现了一个控制非玩家车辆行为的基本控制器：车道跟随、遵守红绿灯、速度限制和交叉路口决策。车辆和行人可以相互察觉和避开。更先进的非玩家车辆控制器将被集成到未来的版本中 [1]。

行人根据特定城镇的导航地图在镇上游荡，相互避开，尽量避开车辆。如果汽车与行人相撞，行人将从仿真世界中删除，并在短暂的时间间隔后在不同的位置生成新的行人。

为了增加视觉多样性，我们在将非玩家角色添加到仿真中时对其外观进行随机化。每个行人都穿着从预先指定的衣柜中随机抽取的一套衣服，并可选择配备以下一种或多种物品：智能手机、购物袋、吉他盒、手提箱、雨伞等。每辆车都是根据特定车型的一组材料随机喷漆的。

我们还实现了各种大气条件和照明条件。它们在太阳的位置和颜色、天空漫射的强度和颜色以及环境遮挡、大气雾、云量和降水量等方面存在差异。目前，CARLA 支持两种照明条件（正午和日落）以及 9 种天气条件（不同的云量、降水量和街道上是否有水坑）。这将实现总共 18 种照明天气组合（为了简洁起见，我们将其称为天气。）

传感器。CARLA 允许灵活配置传感器套件。在撰写本文时，传感器仅限于 RGB 摄像头和提供地面真实深度和语义分割的传感器。如图 2 所示。摄像头的数量及其类型和位置可由用户指定。摄像头参数包括三维位置、相对于汽车坐标系的三维方向、视野和景深。我们的语义分割伪传感器提供了 12 个语义类：道路、车道标线、交通标志、人行道、围栏、标杆、墙、建筑、植被、车辆、行人和其他。

CARLA 支持自动驾驶系统的开发、训练和详细的性能分析。我们使用 CARLA 评估了三种自动驾驶方法。第一种是一种 modular pipeline，它依赖于视觉感知、规划和控制的专用子系统。这种结构符合大多数现有的自动驾驶系统 [21,8]。第二种方法基于通过模拟学习进行端到端训练的深度网络 [4]。这种方法最近引起了新的兴趣 [22, 16, 4]。第三种方法基于通过强化学习进行端到端训练的深度网络 [19]。

3.1 modular pipeline

我们的第一种方法是一个 modular pipeline，它将驾驶任务分解到以下子系统中：(i) 感知；(ii) 规划；(iii) 持续控制。由于没有提供几何地图作为输入，视觉感知成为一项关键任务。局部规划完全依赖于感知模块估计的场景布局。

感知部分使用语义分割来估计车道、道路限制、动态对象和其他危险。此外，还使用分类模型来确定交叉口的接近度。规划器使用基于规则的状态机。持续控制由 PID 控制器执行，该控制器驱动转向、节气门和制动机构。现在我们将更详细地描述这些模块。

感知。我们在这里描述的感知是建立在基于 RefineNet 的语义分割网络上的 [17]。训练网络将图像中的每个像素分为以下语义类别之一：C=froad、sidewalk、lane marking、dynamic object、misscellaneous staticg。该网络使用 CARLA 在训练环境中生成的 2500 张标注图像进行训练。根据道路面积和车道标线，利用网络提供的概率分布来估算车道。

此外，我们使用基于 AlexNet 的二元场景分类器（交叉/无交叉）来估计到达交叉路口的可能性 [15]。这个网络是在两个类之间平衡的 500 幅图像上训练的。

规划器。规划器通过生成一组路径点来实现低级别导航：近期目标状态表示车辆在不久的将来所需的位置和方向。规划器的目标是合成使汽车保持在道路上并防止碰撞的路径点。规划器基于状态机，状态机具有以下状态：(i) 道路跟随，(ii) 左转，(iii) 右转，(iv) 交叉路口向前和 (v) 危险停车。状态之间的转换基于感知模块提供的估计值和全局规划器提供的拓扑信息来执行。路径点连同车辆当前的姿态和速度一起传送给控制器。

持续控制器。我们使用比例-积分-微分（PID）控制器 [6]，因为它简单、灵活，并且对慢响应时间具有相对的鲁棒性。每个控制器接收当前姿态、速度和路径点列表，并分别驱动转向、油门和制动机构。我们的目标巡航速度为 20 公里/小时。

3.2 模仿学习

我们的第二种方法是条件模仿学习，这是一种除了感知输入外还使用高级命令的模拟学习 [4]。该方法利用城镇中人类驾驶员记录的一个驾驶轨迹数据集。The dataset $D = \{hoi; ci; ai\}_{i=1}^n$ consists of tuples, each of which contains an observation oi , a command ci , and an action ai . 这些命令由驾驶员在数据采集过程中提供，并指示他们的意图，类似于转向信号灯。我们使用一组四个命令：沿车道行驶（默认），在下一个十字路口直行，在下一个十字路口左转，在下一个十字路口右转。观察结果是来自前向摄像机的图像。为了提高学习策略的鲁棒性，我们在数据采集过程中加入了噪声。

我们已经收集了大约 14 个小时的驾驶数据用于训练。使用 Adam 优化器对网络进行训练 [14]。为了提高泛化能力，我们进行了数据扩充和删除。

3.3 强化学习

我们的第三种方法是深度强化学习，它基于环境提供的奖励信号训练一个深度网络，没有人类驾驶轨迹。我们使用 A3C[19]。该算法在仿真的三维环境中表现良好，例如赛车 [19] 和三维迷宫中的导航 [19,13,5]。该方法的异步特性使多个线程能够并行运行，这对于深度强化学习的高样本复杂度非常重要。

我们训练 A3C 进行目标导向的导航。在每一次训练中，车辆必须在拓扑规划器的高级命令指导下达到目标。当车辆到达目标时，当车辆与障碍物相撞时，或当时间预算用尽时，事件终止。奖励是五项的加权和：朝目标行驶的速度和距离（正加权）、碰撞（负加权）、与人行道重叠（负加权）、与对面车道重叠（负加权）。

该网络使用 10 个并行线程进行训练，总共进行 1000 万个 simulation steps。因为仿真所带来的计算成本，我们将训练限制在 1000 万个 simulation steps。这相当于以每秒 10 帧的速度连续驾驶 12 天。

4 实验我们评估了三种方法——模块化流水线（MP）、模拟学习（IL）和强化学习（RL），在六种天气条件下，在两个可用城镇中的每一个进行四项越来越困难的驾驶任务。我们按照增加难度的顺序组织任务如下：

- 直线：目的地位于起点正前方，环境中没有动态对象。到目标的平均行驶距离，Town1 为 200 米，Town 2 为 100 米。
- 一个转弯：目的地离出发点只有一个转弯；没有动态物体。到目标的平均行驶距离，Town1 为 400 米，Town 2 为 170 米。
- 导航：不限制目的地相对于起点的位置，无动态物体。到目标的平均行驶距离，Town1 为 170 米，Town 2 为 360 米。
- 存在动态障碍物的导航：与上一个任务相同，但使用动态对象（汽车和行人）。

实验在两个城镇进行。Town1 用于培训，Town2 用于测试。我们考虑六种天气条件进行实验，分成两组。训练用的天气集包括晴天、晴朗的日落、下雨的白天和雨后的白天。测试集的天气是训练集不包含的，包括多云的白天和细雨的日落。

对于一个任务、一个城镇和一个天气集合的每一个组合，都要进行超过 25 次的测试。在每一次测试中，目标是到达指定的目标位置。如果在预定时间内达到目标，则认为事件成功。预定时间为以 10 km/h 的速度沿着最佳路径达到目标所需的时间。违规行为：如在人行道上驾驶或产生碰撞，不会导致事件终止，但会记录和报告。

结果表明了几个一般性结论。总的来说，即使是在最简单的直线驾驶任务中，所有方法的性能都不完美，对于更困难的任务，成功率进一步下降。对新天气的泛化要比对一个新城镇的泛化容易得多。模块化流水线和模拟学习在大多数任务和条件下都能达到同等水平。强化学习相对于其他两种方法表现不佳。现在我们将更详细地讨论这四个关键发现。

四项任务的表现。令人惊讶的是，在训练条件下，即使是在空旷的街道上笔直行驶这一最简单的任务，也没有一种方法能完美地发挥作用。我们认为，这一现象的根本原因是：输入的可变性。训练条件包括四种不同的天气条件。在训练过程中的精确轨迹不会在测试中重现。因此，完美地完成这项任务需要鲁棒的泛化，这对现有的深度学习方法是有挑战的。

对于更高级的任务，所有方法的性能都会下降。在人口稠密的城市环境中的导航任务，两种最好的方法（模块化流水线和模拟学习）在所有条件下的成功率都低于 90%。这些结果清楚地表明，即使在训练条件下，性能也远未达到饱和，并且在新环境下的泛化是一个严重的挑战。

泛化。我们研究两种类型的泛化：对以前没遇到的天气条件和以前没遇到的环境。有趣的是，这两者的结果截然不同。对于模块化流水线和模拟学习来说，“新天气”条件下的性能与训练条件下的性能非常接近，有时甚至更好。然而，推广到一个新城镇对这三种方法都提出了挑战。在两个最具挑战性的导航任务中，当切换到测试城镇时，所有方法的性能下降了很多。这种现象可以解释为这样一个事实，即模型已经在多种天气条件下训练，但只在一个城镇训练。不同天气下的训练结果可以支持对以前没遇到的天

气进行泛化，但对使用不同纹理和 3D 模型的新城镇则不适用。通过在不同的环境中进行训练，这个问题可能会得到改善。总的来说，我们的结果强调了泛化对基于学习的感觉运动控制方法的重要性。

模块化流水线 vs 端到端学习。分析模块化流水线和模拟学习方法的相对性能具有一定的指导意义。令人惊讶的是，在大多数测试条件下，这两种系统的性能非常接近：这两种方法的性能相差不到 10%。这个结论有两个例外：一是模块化流水线在“新天气”条件下比在训练条件下表现更好。这是由于训练和测试天气的特定选择：感知系统恰好在测试天气下表现更好。另一个是，模块化流水线在“新城镇”条件下的导航任务表现不佳，在“新城镇和新天气”下的直线任务表现不佳。这是因为感知算法在新环境的复杂天气条件下系统性地失效。如果感知算法无法可靠地找到可驾驶路径，则基于规则的规划器和经典控制器将无法以一致的方式导航到目的地。因此，如果感知算法正常工作，整个系统工作正常；否则它将完全失败。从这个意义上说，模块化流水线比端到端方法更脆弱。

模仿学习与强化学习。我们现在对比两个端到端训练系统的表现：模仿学习和强化学习。在所有任务中，强化学习训练的表现都比模仿学习的要差。尽管如此，强化学习的训练使用的数据量要大得多：强化学习的数据是驾驶 12 天的，而模仿学习的仅是 14 小时。为什么这一次强化学习表现不佳，而在 Atari 游戏 [18,19] 和迷宫导航 [19,5] 等任务上取得了很好的成绩？一个原因是众所周知强化学习是脆弱的 [12]，并且 it is common to perform extensive task-specific hyperparameter search，例如 Mnih 等人报告的每个环境 50 次试验 [19]。当使用模拟器时，这种 extensive hyperparameter search 变得不可行。我们根据文献证据和迷宫导航的探索性实验选择 hyperparameters。另一种解释是，城市驾驶比以前用强化学习解决的大多数任务更困难。例如，与迷宫导航相比，在驾驶场景中必须处理混乱动态环境中的车辆动态和更复杂的视觉感知。最后，强化学习泛化能力差的原因可能是：与模拟学习相比，强化学习的训练没有数据扩充或规则化。

违规分析。CARLA 支持驾驶策略的细粒度分析。现在，我们将研究三个系统在最困难的任务上的行为：存在动态对象的导航。我们通过以下五种违规行为中任意两种行为之间的平均行驶距离来评价这三个系统：在相反车道上行驶、在人行道上行驶、与其他车辆相撞、与行人相撞和撞击静止物体。

4.2 更多技术支持

交通场景指的是交通参与者在仿真世界中多样的动态行为，通过这些动态行为对运行在其中的自动驾驶车辆进行充分测试。交通场景的丰富性依赖于交通参与者的种类和其能实现的动态行为，CARLA 支持轿车、SUV、客车、卡车、摩托车、自行车、行人等多种动态参与者及锥桶、售货机等多种静态参与者，动态参与者的行为可通过预先定义的场景和在线运行的交通流来控制。

CARLA 中的交通管理器（Traffic Manager）模块可进行场景和交通流的模拟，不过鉴于基于 OpenSCENARIO 格式的场景仿真更通用，我们选用 CARLA 提供的场景运行

器（ScenarioRunner，以下简称 SR）来进行场景的模拟。下面对 SR 的安装和使用进行说明。

01. ScenarioRunner 的安装 ScenarioRunner 是由 CARLA 官方提供的、与 CARLA 配合使用的场景解析和运行工具，支持 CARLA 自定义的 scenario 格式、route 格式和 OpenSCENARIO 格式等多种预定义场景文件的运行，本书主要使用其 OpenSCENARIO 场景运行功能。OpenSCENARIO 目前已经发布 1.2 和 2.0 版本，其中 1.0 和 2.0 版本都在 ScenarioRunner 中得到了支持。

OpenSCENARIO 是德国自动化及测量系统标准协会 ASAM 提供的一个描述动态场景的标准格式，关于 OpenSCENARIO 格式的内容，大家可以看下之前对 OpenSCENARIO 的格式介绍和实例分析。

SR 的安装步骤如下：

(1) 下载源码

SR 的 github 上提供了与 CARLA 版本相配合的 Release 版本 [https://github.com/carla-simulator/scenario_runner/releases]，如 SR0.9.13 与 CARLA 0.9.13 相配合。这是因为 SR 需要使用 PythonAPI 从 CARLA 获取信息并对 CARLA 中的交通参与者、天气等进行控制，如果版本不匹配的话会操作失败。为了获取最新的特性，我们这里使用下载源码的方式进行安装。

大家可以选择将 SR 下载到任何位置，为了方便起见这里下载到 CARLA 文件夹下。

\$ cd /path/to/carla \$ git clone https://github.com/carla-simulator/scenario_runner.git (2) 依赖库安装

进入 scenario_runner 文件夹，并根据其中的 requirements.txt 安装依赖库：

\$ cd scenario_runner \$ sudo apt remove python3-networkx # 若安装过 networkx 则先将其卸载 \$ pip3 install -r requirements.txt 按照以上步骤安装依赖后，若本地 numpy 版本高于 1.20，运行时可能有包含如下字符的报错：.....networkx/readwrite/graphml.py.....module 'numpy' has no attribute 'int'.....。这是因为 requirements.txt 中指定的 networkx 2.2 版本使用了 http://np.int，该用法在 numpy 1.20 版本以上已经不再支持。读者可以根据实际情况安装高版本 networkx 或者低版本的 numpy。

以下两种方法选一即可 #1. 卸载 networkx，并重新安装新版本 \$ pip3 uninstall networkx \$ pip3 install networkx #2. 卸载 numpy，并重新安装低版本 \$ pip3 uninstall numpy \$ pip3 install numpy==1.20 (3) 设置环境变量

为了在运行时能够找到相关的文件，需要设置一些环境变量。打开 /.bashrc 文件，并在其结尾加入如下内容：

```
export CARLA_ROOT=/path/to/carla export SCENARIO_RUNNER_ROOT=$CARLA_ROOT/scenario_runner
export PYTHONPATH=$PYTHONPATH:$CARLA_ROOT/PythonAPI/carla 大家请注意将其中路径修改为自己电脑上的实际路径，然后运行 source /.bashrc 使设置生效。
```

至此用于运行 OpenSCENARIO 1.0 文件（以下简称 xosc 文件）的安装工作都已完成，大家可尝试按照下一节的方法运行相关文件。若想运行 OpenSCENARIO 2.0 文件

(以下简称 osc 文件)，还需要进行如下操作。

(4) 安装 OpenSCENARIO 2.0 相关依赖

```
# 安装 JDK $ sudo apt install openjdk-17-jdk # 安装 Antlr $ curl -O https://www.antlr.org/download/antlr-4.10.1-complete.jar $ sudo cp antlr-4.10.1-complete.jar /usr/local/lib/ $ pip3 install antlr4-python3-runtime==4.10 打开 /.bashrc 文件，并在其结尾加入如下内容，然后运行 source /.bashrc 使设置生效。
```

```
export CLASSPATH=".:./usr/local/lib/antlr-4.10.1-complete.jar:$CLASSPATH" alias antlr4='java -jar ./usr/local/lib/antlr-4.10.1-complete.jar' alias grun='java org.antlr.v4.gui.TestRig' xport CARLA_ROOT= export SCENARIO_RUNNER_ROOT=$CARLA_ROOT/scenario_runner export PYTHONPATH=$PYTHONPATH:$CARLA_ROOT/PythonAPI/carla 02. 运行 OpenSCENARIO 文件 使用 SR 运行 xosc/osc 文件的步骤十分简单，首先启动 CARLA，然后运行 SR 并指定 xosc/osc 文件即可：
```

(1) 启动 CARLA:

```
$ cd /path/to/carla $ ./CarlaUE4.sh (2) 配置 ego 车
```

实际测试时应由被测算法控制 ego 车，此处为了进行演示，通过 SR 自带的 manual_control.py 为 ego 车配置自动驾驶：

```
$ cd /path/to/scenario_runner # 加载 xosc 文件示例时使用 $ python3 manual_control.py -a # 加载 osc 文件示例时使用 $ python3 manual_control.py -a --rolename ego_vehicle 需要注意的是，manual_control.py 根据 rolename 查找 ego 车辆并为其配置自动驾驶，默认 ego 车辆的 rolename 为“hero”，在下面的 xosc 文件示例中 ego 车辆的 rolename 恰好为“hero”，故无需配置“--rolename”参数，而 osc 文件示例中 ego 车辆的 rolename 为“ego_vehicle”，从而需要指定“--rolename”。
```

(3) 运行 ScenarioRunner

```
$ cd /path/to/scenario_runner # 运行 xosc 文件示例 $ python3 scenario_runner.py --output --openscenario srunner/examples/FollowLeadingVehicle.xosc # 运行 osc 文件示例 $ python3 scenario_runner.py --output --openscenario2 srunner/examples/cut_in_and_slow_range.osc 运行上述命令后，可以在 CARLA 渲染窗口中观察到地图根据 xosc 文件中定义变更，同时生成了 ego 车和其前方的障碍车。
```

4.3 总结与展望

4.3.1 总结

自动驾驶汽车技术发展需科学测试体系支持。虚拟仿真测试是解决封闭和开放道路测试成本和效率问题的有效方法，尽管理论和技术体系待完善。数字虚拟仿真在生成危险场景方面具有明显优势，是提升测试安全性和可靠性的关键，正成为研究焦点。

现有的自动驾驶危险场景生成方法包括专家经验法、自动驾驶数据提取法、危险场景衍生法和基于强化学习的自生成法。专家经验法主观且难以覆盖所有场景，自动驾驶数据提取法缺乏动态交互能力，危险场景衍生法可能产生不真实的轨迹。这些方法多限于学术研究，难以应用于工程实践。基于强化学习的自生成法能实时修正场景状态，生成定制化场景库，适合高级别自动驾驶测试，成为研究热点。但现有方法可能产生不自然或重复场景，缺乏泛化能力。本项目旨在设计一种综合自然性、对抗性和多样性的统一生成策略，解决现有方法在实际应用中的不足。

为了探索自动驾驶汽车的性能极限，评估自动驾驶系统的安全性、可靠性和可信度，这需要精确且完整地描述自动驾驶的性能边界。现有的研究未使用真实的自动驾驶模型和算法进行性能边界的挖掘，这可能导致缺乏现实性、误导性结果和可迁移性问题。同时，现有的研究仅考虑周围车辆产生危险行为会限制对整体风险的评估。因为行人和环境因素可能引入新的风险和挑战，需要在危险场景生成中进行考虑。忽略这些因素可能导致对风险的低估，无法全面评估自动驾驶系统在复杂和多变环境中的行为和决策能力。当前研究仅仅利用背景车辆的初始运行条件（位置、速度、航向角等）来构造驾驶场景的参数空间，因此更加高维的参数空间应该被定义以精细化和全面地表征自动驾驶性能边界。当处理高维参数空间和昂贵的自动驾驶仿真评价时，通用的启发式智能优化算法往往将不再适用，能进行有效求解的优化算法非常有限。在这种情况下，非常有必要借助代理建模、最优计算资源分配、样本填充策略等技术设计相应的随机或鲁棒仿真优化算法，这也是本项目研究的关键点。

作为当前世界最先进的自动驾驶方案，端到端自动驾驶具有较强的感知性能和泛化能力，构建的自动驾驶系统更加简单智能。然而现有的端到端自动驾驶决策方法还存在以下问题：1) 端到端自动驾驶模型依赖训练数据，面对复杂和危险场景时可能做出不合理决策；2) 端到端自动驾驶依赖深度神经网络，这些网络的黑盒特性使得决策过程难以解释，限制了系统行为的理解和调试；3) 端到端自动驾驶模型从图像或点云直接输出控制信号，训练中难以全面理解复杂交通环境，无法考虑车辆互动、行人、信号灯等多重因素，导致决策缺乏适应性和灵活性。为端到端自动驾驶设计一个富含驾驶语义信息的输入，并利用先进的神经网络架构突破以往自动驾驶的性能边界是本研究的终极目标。

本研究紧密围绕自动驾驶场景生成与安全评估的核心问题，深入探索并提出 ChatScene 与 ASIL-Gen 协同解决方案，成功构建起从场景高效生成到精准风险评估的完整技术体系，为自动驾驶测试领域带来了创新性突破。在场景生成层面，ChatScene 框架展现出独特的技术优势。其创新性地融合 Scenic 语言、深度强化学习与 GPT-4o 模型，实现了

场景生成模式的多元化与智能化。在固定场景模式下，基于 Scenic 语言预定义的 8 类基准场景，涵盖城市主干道、高速公路、乡村道路等多种典型交通环境，通过深度强化学习中的 SAC 算法，以碰撞避免、车道保持、速度优化等为核心奖励函数，对场景中对抗行为参数进行精细化调整与优化，从而模拟出高度真实、复杂多变的交通参与者交互行为，为自动驾驶系统提供标准化、规范化的测试场景模板。在动态模式方面，借助 GPT-4o 强大的自然语言理解与代码生成能力，用户仅需输入自然语言描述，如“暴雨天气下，自我车辆在无信号灯十字路口左转，对向车辆突然加速直行”，系统便能快速解析语义信息，并将其转化为符合 CARLA 0.9.13 规范的场景代码。通过设定详细的约束条件，包括道路类型、车辆动力学参数、交通规则等，生成多样化、贴近真实驾驶场景的测试用例，相比传统基于 LiDARGen 的场景生成方法，动态模式的生成效率提升了 107 倍，极大地丰富了自动驾驶测试场景的多样性与灵活性，显著提高了测试效率。

在安全评估领域，ASIL-Gen 发挥了关键作用。该模块通过 NSGA-II 多目标优化算法与基于 ISO 26262 标准的量化评估模型，实现了场景的优化筛选与科学分类。在场景生成阶段，基于基础场景模板，通过脚本自动化生成 13 类场景变体，包括动态障碍物交叉、无信号路口冲突、恶劣天气下的跟车等复杂情况，并通过调整障碍物位置、速度、交通流量等参数，形成海量多样化的测试场景。在此基础上，利用 NSGA-II 算法，以碰撞概率与场景复杂度作为双目标函数进行优化。碰撞概率通过精确预测车辆、行人等交通参与者的运动轨迹进行计算，场景复杂度则综合考虑交通参与者数量、道路拓扑结构、环境干扰因素等多个指标，从而搜索出帕累托前沿，筛选出既具有高风险价值又具备代表性的场景。实验数据显示，通过 ASIL-Gen 优化后，高风险 ASIL-D 等级场景占比从随机搜索的 5% 提升至 12%，经过进一步优化后更是提升至 15%，显著增强了场景筛选的精准性与安全评估的科学性。

在实验验证环节，本研究取得了一系列具有说服力的成果。基于 ChatScene 训练的 RL 模型在行人横穿场景测试中，与基线模型进行多次重复对比实验，通过精确统计碰撞次数，结果表明其碰撞率降低了 32%，充分验证了 ChatScene 与 ASIL-Gen 协同方案对提升自动驾驶系统鲁棒性的有效性。同时，在场景生成效率、安全评估准确性等方面的实验数据，均有力地证明了该方案在自动驾驶技术安全测试中的巨大应用价值，为自动驾驶技术的安全测试开辟了新的技术路径与研究方向。

安全模型，英伟达的安全力场（SFF）和 Mobileye 的责任敏感安全（RSS）等对决策来说是有可解释性的数学模型。这项工作从头开始实现 SFF，替代未公开的英伟达源代码，并将其与 CARLA 开源模拟器集成。使用 SFF 和 CARLA，提出了一个车辆声明集合的预测器，并以此提出一种综合驾驶策略，无论在通过动态交通时遇到什么安全条件，其都能持续运行。该策略没有针对每种情况制定单独的规划，但利用安全潜能，目的是将类人的驾驶融入交通流中。

责任敏感安全（RSS）将自车的危险时间 t 与纵向/横向的危险阈值时间 t_{long}/t_{lat} 进行比较。如果达到阈值，RSS 判断为危险情况，并根据纵向或横向加速度对速度的限制做出适当的响应。换句话说，阈值可以用轨迹集的多边形表示。如果自车和其他道路使

用者之间的轨迹集相交，RSS 会选择以下三个决策中的一个来恢复安全状态：刹车，继续前进或开车离开。

安全力场 (SFF) 表示，如果参与者遵循安全程序（这是一系列控制策略），量化风险的安全潜能 ρ_{AB} 不会再增加，因此可以保证参与者最终不会造成不安全的情况。这可以通过安全潜能的链式法则从数学上证明。简而言之，RSS 的方法是最小化参与者声称集合之间的交集，这是每个参与者安全程序产生轨迹的联合。

英特尔发布了一个名为 ad-rss-lib 的开源库，该库部分实现了 RSS。此外，NVIDIA 还提供了一个名为 DriveWorks SDK 的软件开发工具包，其中包括针对经批准用户的 SFF 实现。Intel ad-rss-lib 没有涵盖其论文的全部范围，但它提供了 Python 绑定和 CARLA 集成。然而，NVIDIA DriveWorks SDK 是一个非公开 IP，它的实现是为了与配备 NVIDIA DRIVE OS 的 NVIDIA DRIVE 平台集成，因此研究人员很少使用它。在 Intel 和 NVIDIA 建议的将 RSS 和 SFF 与现有自动驾驶系统集成的基本示例架构中，RSS 和 SFF 扮演着最后的角色，通过重写规划子系统的决策来防止自动驾驶车辆发生碰撞。

在 SFF 实施中使用声明集合和安全潜能的概念，声明集合就是安全程序（驾驶策略）获取的轨迹之联合，安全潜能是两个参与者的声明集合之间相交测度及其负梯度。该文提出了一种将 SFF 集成到规划子系统中的方法，制定一种类似人类的驾驶策略，无论在安全或不安全的条件下都能始终如一地运行，最终尽量不阻碍顺畅的交通流。

Intel RSS 或 NVIDIA SFF 作为与现有子系统协调的附加模块。它接收来自感知子系统的世界环境数据和来自规划子系统的机动决策。为了自车的安全，作为上层限制器，它可以推翻接收的决策，并将限制的决策传递给驾驶子系统。如图是具有 RSS 或 SFF 安全模型的基本示例架构：灰色是现有子系统部分，蓝色/绿色是附加模块的 RSS/SFF 实现部分。

4.3.2 展望

尽管本研究在自动驾驶场景生成与安全评估领域取得了一定成果，但面对自动驾驶技术快速发展的需求与挑战，该领域仍存在诸多亟待深入探索的方向，未来研究可从以下多个维度展开：

深化多模态数据融合：目前，自动驾驶场景生成主要依赖有限的数据源，未来研究将进一步深化多模态数据融合。除了继续探索激光雷达、摄像头数据的深度融合外，还将引入毫米波雷达数据，利用其在恶劣天气下的稳定探测性能，提升场景中目标检测与跟踪的准确性；同时融合高精地图数据，将地图中的道路属性、交通标志、车道线等信息与传感器数据相结合，构建更贴近真实驾驶环境的场景生成模型。通过多模态数据的协同处理与特征融合，不仅能够提升场景的物理真实性，还能增强场景的语义准确性，使生成的场景更符合实际驾驶过程中的复杂情况，为自动驾驶系统提供更具挑战性与真实性的测试环境。

拓展 ASIL 分类标准：现有的 ASIL 分类标准主要基于静态的交通场景因素，未来将结合车路协同、交通流理论等前沿技术，对其进行拓展与完善。将 V2X 通信状态纳

入评估体系，考虑车辆与车辆、车辆与基础设施之间的信息交互对驾驶安全的影响；同时引入交通拥堵态势分析，结合交通流模型，评估不同拥堵程度下自动驾驶系统面临的风险。此外，还将探索驾驶员行为模型与 ASIL 分类的结合，分析人类驾驶行为的不确定性对自动驾驶系统安全的潜在影响，从而完善高风险场景的识别维度，使 ASIL 分类标准更全面、准确地反映自动驾驶系统在复杂交通环境下的安全风险状况。

探索模型轻量化与实时化：随着自动驾驶技术向边缘设备的不断渗透，算力受限成为制约实时在线评估的关键因素。未来研究将致力于探索 ChatScene 与 ASIL-Gen 算法架构的轻量化与实时化优化。一方面，通过模型压缩技术，如剪枝、量化等方法，减少模型参数与计算量；另一方面，采用高效的算法优化策略，如改进的神经网络结构、并行计算技术等，提高算法执行效率。针对边缘设备的硬件特性，开发专用的轻量化模型，使其能够在资源有限的条件下快速生成场景并进行安全评估，推动自动驾驶仿真测试从离线分析向实时在线评估方向发展，为自动驾驶车辆的实时决策与安全监控提供有力支持，加速自动驾驶技术的落地应用进程。

强化人机协同测试模式：考虑到自动驾驶系统最终服务于人类出行，未来可探索将人类驾驶员的行为与决策纳入测试体系，构建人机协同的测试模式。通过收集和分析人类驾驶数据，模拟不同驾驶风格、驾驶习惯的人类驾驶员与自动驾驶系统的交互场景，评估自动驾驶系统在人机共驾环境下的安全性与适应性。同时，研究如何利用人类驾驶员的经验与直觉，辅助自动驾驶系统进行决策，实现人机优势互补，进一步提升自动驾驶系统的可靠性与用户接受度。

开展跨地域与跨文化场景研究：目前的研究主要基于特定地域的交通规则与驾驶习惯，未来将开展跨地域与跨文化场景研究。收集不同国家、不同地区的交通数据，分析其交通规则、驾驶文化的差异，构建多样化的跨地域场景库。通过在这些场景下对自动驾驶系统进行测试与评估，确保自动驾驶技术在全球范围内的适用性与安全性，推动自动驾驶技术的全球化发展。

4.4 参考文献

Cheng Bo, Lin Qingfeng, Song Tianjia, Cui Yongwei, Wang Limian, Kuzumaki Seigo. Analysis of Driver Brake Operation in Near-Crash Situation Using Naturalistic Driving Data[J]. International Journal of Automotive Engineering, 2011, 2(4).

Mckinsey & Company. Autonomous driving's future: Convenient and connected [EB/OL]. (2023-01-01)[2023-03-05].<https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenientand-connected>

ChatScene: Knowledge-Enabled Safety-Critical Scenario Generation for Autonomous Vehicles Jiawei Zhang, Chejian Xu, Bo Li; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 15459-15469

张开元. FOT 数据库危险驾驶工况数据的自动处理方法研究 [D]:[硕士学位论文]. 上海: 同济大学, 2016.

Fitch G M, Hanowski R J. Using Naturalistic Driving Research to Design, Test and Evaluate Driver Assistance Systems[M]. Springer London, 2012.

朱西产, 张佳瑞, 马志雄. 安全切入场景下的驾驶人初始制动时刻分析 [J]. 中国公路学报, 2019, 32(6).

杨敏明, 王雪松, 朱美新. 基于自然驾驶实验的驾驶行为研究 [J]. 交通与运输, 2017, 33(03):7-9.

VL Neale, SG Klauer, TA Dingus, et al. The 100-Car Naturalistic Driving Study, Phase I-Experimental Design. Behavior, 2002.

TA Dingus, SG Klauer, VL Neale, et al. The 100-Car Naturalistic Driving Study, Phase II-Results of the 100-Car Field Experiment. Behavior, 2006.

Characterizing Underground Utilities[J]. Shrp Report, 2009.

Integrated Vehicle-Based Safety Systems (IVBSS) Phase I Interim Report[J]. 2008.

Sayer J, Leblanc D, Bogard S, et al. Integrated Vehicle-Based Safety Systems(IVBSS) light vehicle platform field operational test data analysis plan[J]. HeavyDuty Trucks, 2009.

Sayer J R, Leblanc D J, Bogard S E. Integrated Vehicle-Based Safety Systems(IVBSS) Third annual report[J]. Scottish Journal of Theology, 2013, 22:382-383.

Administration NHTS. Integrated Vehicle-Based Safety Systems: Light VehicleField Operational Test, Key Findings Report[J]. Annals of Emergency Medicine, 2011, 58(2):205-206.

Benmimoun M, Fahrenkrog F, Zlocki A , et al. Incident detection based on vehicleCAN-data within the large scale field operational test[J]. 2011.

Benmimoun M, Eckstein L. Detection of Critical Driving Situations for NaturalisticDriving Studies by Means of an Automated Process[J]. bowen publishing, 2014:11-21.

Victor T, Jonas BÄrgman, Gellerman H, et al. Sweden-Michigan NaturalisticField Operational Test (SeMiFOT) Phase 1: Final Report[J]. 2010.

刘颖. 行人自动紧急制动系统测试方法研究 [D]. 同济大学, 2014.

NAO Engineering, Safety & Restraints Center, Crash Avoidance Department, “44-Crashes”, General Motors Corporation, Version 3.0, January 1997.

Crash Avoidance Metrics Partnership, “Enhanced Digital Mapping Project -FinalReport”. U.S. Department of Transportation, National Highway Traffic Safety Administration, November 2004.

W.G. Najm, B. Sen, J.D. Smith, and B.N. Campbell, “Analysis of Light VehicleCrashes and Pre-Crash Scenarios Based on the 2000 General Estimates System”.DOT-VNTSC-NHTSA-02-04, DOT HS 809 573, February 2003.

Najm W G, Smith J D, Yanagisawa M. Pre-Crash Scenario Typology for CrashAvoidance Research[J]. Dot Hs, 2007:767.

曾宇凡, 朱西产, 马志雄, 等. 基于真实事故和自然驾驶场景大的车辆-骑车人危险工况的统计分析 [J]. Infats Proceedings of the 14th International Forum ofAutomotive Traffic Safety, 2017(01):1-9.

苏江平, 陈君毅, 王宏雁, 等. 基于中国危险工况的行人交通冲突典型场景提取与分析 [J]. 交通与运输 (学术版), 2017(01):216-221.

孟琳, 朱西产, 孙晓宇, 等. 真实交通危险工况下驾驶员转向避撞相关因素分析 [J]. 汽车技术, 000(6):59-62.

李霖, 朱西产, 马志雄. 驾驶员在真实交通危险工况中的制动反应时间 [J]. 汽车工程, 2014(10):1225-1229.

吴斌, 朱西产, 沈剑平, 等. 基于自然驾驶研究的直行追尾危险场景诱导因素分析 [J]. 同济大学学报: 自然科学版, 2018, 46(09):96-103.

孙招凤. CAN 总线网络报文标识符编码研究 [J]. 导弹与航天运载技术, 2009(02):34-39.

范云锋, 刘博, 郑益凯. 一种基于三次样条曲线的目标航迹拟合与插值方法研究 [J]. 数字技术与应用, 2019, 37(03):128-129.

陈华. 面向智能辅助驾驶系统的驾驶员行为分析与建模 [D].

Thiemann C, Treiber M, Kesting A. Estimating Acceleration and Lane-ChangingDynamics from Next Generation Simulation Trajectory Data[J]. TransportationResearch Record Journal of the Transportation Research Board, 2008, 2088(2088):90-101.

薛薇. SPSS 统计分析方法及应用 [M]. 北京: 电子工业出版社, 2004.

杨学兵, 张俊. 决策树算法及其核心技术 [J]. 计算机技术与发展, 2007, 17(1):43-45.

曾星, 孙备, 罗武胜, 等. 基于深度传感器的坐姿检测系统 [J]. 计算机科学, 2018, v.45(07):243-248.

程娟, 陈先华. 基于梯度提升决策树的高速公路行程时间预测模型 (英文) [J].Journal of Southeast University (English Edition), 2019, 35(03):393-398.79

段文强. 基于用户行为序列的网络购买行为预测 [D]. 江西财经大学, 2019.

Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System[J]. 2016.

师圣蔓. 基于机器学习的网络流量预测与应用研究 [D]. 2019.

靳小波. 基于机器学习算法的文本分类系统 [D]. 西北工业大学.

KE G, MENG Q, FINLEY T, et al. Lightgbm: A highly efficient gradient boosting decision tree[C] // 31st Conference on Neural Information Processing Systems(NIPS 2017). California, USA,2017:3146-3154.

刘彧祺, 张智斌, 陈昊昱, et al. 基于 XGBoost 集成的可解释信用评分模型 [J]. 数据通信, 2019(3).

Zhang Y, Xue J R, Zhang G, el al. A multi-feature fusion based traffic lightrecognition algorithm for intelligent vehicles[C]. Proceeding of the 33rd ChineseControl Conference. IEEE,2014:4924-4929.

Ma C, Xue J, Liu Y, et al. Data-Driven State-Increment Statistical Model and ItsApplication in Autonomous Driving[J]. Intelligent Transportation Systems IEEETransactions on, 2018, 19(12):3872-3882.

致谢

行文至此，落笔为终，回首这段撰写本科论文的时光，心中满是感慨与感激。在此，我想用最真挚的文字，向所有帮助过我的人表达深深的谢意。首先，我要向我的论文导师致以最崇高的敬意与最诚挚的感谢。从论文选题的迷茫，到 ChatScene 框架设计思路的梳理；从 ASIL-Gen 优化算法的探讨，到实验结果分析的困惑，老师始终以渊博的专业知识、严谨的治学态度和耐心细致的指导，为我拨开迷雾。每一次组会的讨论，每一份修改意见的批注，都凝聚着老师的心血。老师不仅教会我学术研究的方法，更用实际行动诠释了精益求精的科研精神，这些都将成为我未来学习和工作的宝贵财富。感谢学院的各位老师，在课堂上，你们生动的讲解让我对自动驾驶领域有了更深入的理解；在论文开题、中期检查等环节，你们提出的建设性意见，帮助我不断完善研究内容。特别感谢实验室的师兄师姐和同学们，在我遇到技术难题时，你们慷慨分享经验，陪我调试 CARLA 仿真平台、优化 NSGA-II 算法代码；在我倍感压力时，你们的鼓励和陪伴，让我重拾信心。与你们一起奋斗的日子，是我本科生涯中最难忘的回忆。还要感谢我的家人，是你们在背后默默的支持，让我能够心无旁骛地投入论文研究。每当我因实验不顺利而焦虑时，是你们的理解与安慰，给予我温暖与力量；你们的关爱与鼓励，是我前行路上最坚实的后盾。最后，感谢参与本研究的所有人员，以及为论文提供数据和技术支持的机构。虽然论文仍存在不足之处，但这段经历让我收获颇丰。未来，我将带着这份感恩之心，继续在学术道路上探索前行。

```
%!TEX root = ../csuthesis_main.tex %
```

```
% 无章节编号
```

附录 A 附录代码

附录部分用于存放这里用来存放不适合放置在正文的大篇幅内容、典型如代码、图纸、完整数学证明过程等内容。

A.1 chatscene

Step 1: Setup conda environment

```
conda create -n chatscene python=3.8 conda activate chatscene Step 2: Clone this git repo  
in an appropriate folder
```

```
git clone git@github.com:javyduck/ChatScene.git Step 3: Enter the repo root folder and  
install the packages:
```

```
cd ChatScene pip install -r requirements.txt pip install decorator==5.1.1 pip install -e . (you  
can ignore the error after installing the decorator)
```

Step 4: Install the Scenic package:

```
cd Scenic python -m pip install -e . Step 5: Download our CARLA_0.9.13 and extract it  
to your folder.
```

Step 6: Run sudo apt install libomp5 as per this git issue.

Step 7: Add the python API of CARLA to the PYTHONPATH environment variable. You
can add the following commands to your /.bashrc:

```
export CARLA_ROOT=path/to/your/carla export PYTHONPATH=$PYTHONPATH :$CARLA_ROOT  
0.9.13-py3.8-linux-x86_64.egg export PYTHONPATH=$PYTHONPATH :$CARLA_ROOT/PythonAPI  
export PYTHONPATH=$PYTHONPATH :$CARLA_ROOT/PythonAPI/carla export PYTHON-  
PATH=$PYTHONPATH :$CARLA_ROOT/PythonAPI Then, do source /.bashrc to update  
the environment variable.
```

1. Desktop Users Enter the CARLA root folder, launch the CARLA server and run our
platform with

```
# Launch CARLA ./CarlaUE4.sh -prefernvidia -windowed -carla-port=2000 2. Remote  
Server Users Enter the CARLA root folder, launch the CARLA server with headless mode, and  
run our platform with
```

```
# Launch CARLA ./CarlaUE4.sh -prefernvidia -RenderOffScreen -carla-port=2000 (Optional)  
You can also visualize the pygame window using TurboVNC. First, launch CARLA with  
headless mode, and run our platform on a virtual display.
```

```
# Launch CARLA ./CarlaUE4.sh -prefernvidia -RenderOffScreen -carla-port=2000
```

```
# Run a remote VNC-Xserver. This will create a virtual display "8". /opt/TurboVNC/bin/vnc-  
server :8 -noxstartup You can use the TurboVNC client on your local machine to connect to the
```

virtual display.

```
# Use the built-in SSH client of TurboVNC Viewer /opt/TurboVNC/bin/vncviewer -via
user@host localhost:n

# Or you can manually forward connections to the remote server by ssh -L fp:localhost:5900+n
user@host # Open another terminal on local machine /opt/TurboVNC/bin/vncviewer localhost::fp where user@host is your remote server, fp is a free TCP port on the local machine, and n is the display port specified when you started the VNC server on the remote server ("8" in our example).
```

ChatScene In ChatScene, we ensure a fair comparison with the baselines by using the same eight scenarios, sampling five behaviors for each scenario from the database. The corresponding generated complete Scenic files, with some modifications, have been provided in safebench/scenario/scenario_data/scenic_data (with some manual modifications to use the same fixed 10 routes for the ego agent to ensure fair comparison with the baselines).

The ego agent is controlled by a default RL model, while the surrounding adversarial agent is controlled by Scenic.

The agent configuration is provided in safebench/agent/config/adv_scenic.yaml. By default, it loads a pretrained RL model from Safebench-v1.

Modes in ChatScene: train_scenario: Select the most challenging scenes for the same behavior under the same scenario.

Configuration can be found in safebench/scenario/config/train_agent_scenic.yaml.

The sample_num = 50, opt_step = 10, select_num = 2 settings in the file mean we sample 50 scenes and select the 2 most challenging ones for evaluation. The default setting is to choose scenes that lead to a collision of the ego agent and provide the lowest overall score. We optimize the range of parameters, like speed, every 10 steps based on collision statistics from previously sampled scenes.

Example command for optimizing the scene:

```
python scripts/run_train.py --agent_cfg=adv_scenic.yaml --scenario_cfg=train_scenario_scenic.yaml
--mode train_scenario --scenario_id 1 Use the following command if you are using a TurboVNC
client on your local machine to connect to the virtual display:
```

```
DISPLAY=:8 python scripts/run_train.py --agent_cfg=adv_scenic.yaml --scenario_cfg=train_scenario_scenic.yaml
--mode train_scenario --scenario_id 1 The IDs for the final selected scenes will be stored in
safebench/scenario/scenario_data/scenic_data/scenario_1/scenario_1.json.
```

train_agent: Train the agent based on the selected challenging scenes:

```
python scripts/run_train.py --agent_cfg=adv_scenic.yaml --scenario_cfg=train_agent_scenic.yaml
--mode train_agent --scenario_id 1 We have a total of 10 routes for each scenario. We use the
first 8 for training and the last 2 for testing (route IDs: [0,1,2,3,4,5,6,7]). The configuration,
including scenario_1.json, will train the agent based on the most challenging scenes (the ones
```

leading to a collision of the ego agent).

eval: Evaluate the trained agent on the last 2 routes (route IDs: [8,9]), the test_epoch is for loading a finetuned model after a specific training epoch:

```
python scripts/run_eval.py --agent_cfg=adv_scenic.yaml --scenario_cfg=eval_scenic.yaml  
--mode eval --scenario_id 1 --test_epoch -1 The -1 here is for loading our provided fine-tuned  
agent in each scenario based on our Scenic scenarios in safebench/agent/model_ckpt/adv_train/sac/scenic/sc-  
001.torch.
```

Dynamic Mode The above part ensures using the same scenario and routes for fair comparison with baselines. However, ChatScene can generate scenarios and scenes freely without any constraints. Simply provide a text description, such as "The ego vehicle is driving on a straight road; the adversarial pedestrian suddenly crosses the road from the right front and suddenly stops in front of the ego." is enough for the training. We are currently integrating our database with GPT-4o for generating more diverse scenarios based on our pre-built retrieval database, and will upload both soonly.

Please first install openai and sentence_transformers packages following the requirements.

Put your description under file retrieve/scenario_descriptions.txt

run python retrieve.py to get the corresponding scenic code under safebench/scenario/sce-
nario_data/scenic_data/dynamic_scenario

Then, for running the dynamic scenarios, just replace the run_train.py or run_eval.py with run_train_dynamic.py or run_eval_dynamic.py, and use dynamic_scenic.yaml (please specify your settings there), an exmaple could be:

```
python scripts/run_train_dynamic.py --agent_cfg=adv_scenic.yaml --scenario_cfg=dynamic_scenic.yaml  
--mode train_scenario Integrate GPT-4o with our retrieval database (v1) and commit to the dy-  
namic mode. Some mechanisms have been changed based on the previous version to incorporate  
more adversarial behavior, geometry, and spawn point definitions. Currently, it is still in beta.  
If you encounter any problems, please submit an issue, and I will address potential errors in the  
new retrieval pipeline. Some snippets are still under cleaning of the updated framework (i.e.,  
incorporating GPT-4o to generate more diverse scenarios), the new retrieve database v2 will be  
pushed based on the new design. Finetune an LLM for generating snippets end-to-end based  
on the data constructed from our database.
```

A.2 ASIL-Gen

ASIL-Gen: Automotive Scenario Generation and ASIL Classification Toolkit A toolkit for generating automotive scenarios, evaluating their safety levels (ASIL), and comparing optimization algorithms (NSGA-II vs. Random Search) for scenario selection.

Overview This repository contains tools for:

Generating variations of driving scenarios Executing scenarios in CARLA simulator Se-

lecting critical scenarios using NSGA and Random Search algorithms Determining Automotive Safety Integrity Levels (ASIL) of selected scenarios Analyzing results with statistical methods Prerequisites CARLA Simulator: Install CARLA from carla-simulator/carla. Scenario Runner: Install the CARLA Scenario Runner from carla-simulator/scenario_runner. Setup Clone this repository:

```
git clone https://github.com/Fizza129/ASIL-Gen.git
```

Install Python dependencies (if required): Ensure , , (for NSGA-II), and other standard libraries are installed.numpy pandas deap

Usage 1. Running Scenarios Pre-generated Scenarios:

Extract .Scenario_Dataset/Scenario_Dataset_1-4.zip Copy scenario files to ..pyscenario_runner/srunner Copy configuration files to ..xmlscenario_runner/srunner/examples/ Follow Scenario Runner documentation to execute scenarios. Custom Scenarios: Use scripts in to generate new variations. Example: Scenario Generation Scripts/

python script_change_lane.py # Generates new lane-change variations 2. Scenario Selection NSGA-II Optimization: Run scripts in the folder on scenario execution results: NSGA/ python "NSGA/NSGA_choice.py" Random Search: Execute scripts in the folder: The selected scenarios will be saved in JSON format. Random Search/ python "Random Search/random_search_choice.py" 3. ASIL Classification Use scripts in the folder to compute ASIL levels (A/B/C/D/QM) for selected scenarios: ASIL/ python "ASIL/ASIL.py" # Calculates ASIL levels python "ASIL/ASIL_percentages.py" # Calculates ASIL distribution percentages 4. Statistical Comparison (NSGA vs. Random Search) Run Mann-Whitney U Test scripts in : Mann Whitney Test/ python "Mann Whitney Test/Mann Whitney and Effect Size.py" Repository Structure ASIL-Gen/ └── ASIL/ # ASIL classification and percentage calculation └── Mann Whitney Test/ # Statistical tests for comparing NSGA and Random Search └── NSGA/ # NSGA-II optimization for scenario selection └── Random Search/ # Random Search-based scenario selection └── Scenario Dataset/ # Pre-generated scenario variations (Python + XML) └── Scenario Generation Scripts/ # Scripts to generate new scenario variations └── Scenario Results/ # Output results Scenario Types The repository contains 13 unique scenarios:

Lane change Cut-in with static vehicle Dynamic object crossing Following leading vehicle Following leading vehicle with obstacle Hazard at side lane No signal junction crossing Opposite vehicle running red light Other leading vehicle Parked obstacle Parking crossing pedestrian Vehicle opens door Vehicle turning right Results The folder contains the outputs from executed scenarios. Scenario Results

Notes Dataset Usage: The dataset contains 1,000 variations for each unique scenario. Extract and copy files as instructed. Scenario Dataset Custom Generation: Modify scripts in to create new scenario variations. Scenario Generation Scripts/

A.3 scene_runner

```
''' Date: 2023-01-31 22:23:17 LastEditTime: 2023-03-07 12:28:17 Description: Copyright (c) 2022-2023 Safebench Team

This work is licensed under the terms of the MIT license. For a copy, see <https://opensource.org/licenses/MIT>

import copy import os import json import glob import random
import numpy as np import carla import pygame from tqdm import tqdm
from safebench.gym_carla.env_wrapper import VectorWrapper from safebench.gym_carla.envs.render import BirdeyeRender from safebench.gym_carla.replay_buffer import RouteReplayBuffer, PerceptionReplayBuffer
from safebench.agent import AGENT_POLICY_LIST from safebench.scenario import SCENARIO_POLICY_LIST
from safebench.scenario.scenario_manager.carla_data_provider import CarlaDataProvider
from safebench.scenario.scenario_data_loader import ScenarioDataLoader, ScenicDataLoader
from safebench.scenario.tools.scenario_utils import scenario_parse, scenic_parse
from safebench.util.logger import Logger, setup_logger_kwargs from safebench.util.metric_util import get_route_scores, get_perception_scores from safebench.util.scenic_utils import ScenicSimulator
class ScenicRunner: def __init__(self, agent_config, scenario_config): self.scenario_config = scenario_config self.agent_config = agent_config
    self.seed = scenario_config['seed'] self.exp_name = scenario_config['exp_name'] self.output_dir = scenario_config['output_dir'] self.mode = scenario_config['mode'] self.save_video = scenario_config['save_video']
    self.render = scenario_config['render'] self.num_scenario = scenario_config['num_scenario']
    self.fixed_delta_seconds = scenario_config['fixed_delta_seconds'] self.scenario_category = scenario_config['scenario_category']
    # continue training flag self.continue_agent_training = scenario_config['continue_agent_training']
    self.continue_scenario_training = scenario_config['continue_scenario_training']
    # apply settings to carla self.client = carla.Client('localhost', scenario_config['port']) self.client.set_time(self.world = None self.env = None
    self.env_params = 'auto_ego': scenario_config['auto_ego'], 'obs_type': agent_config['obs_type'],
    'scenario_category': self.scenario_category, 'ROOT_DIR': scenario_config['ROOT_DIR'], 'warm_up_steps': 9, # number of ticks after spawning the vehicles 'disable_lidar': True, # show bird-eye view lidar or not 'display_size': 128, # screen size of one bird-eye view window 'obs_range': 32, # observation range (meter) 'd_behind': 12, # distance behind the ego vehicle (meter) 'max_past_step': 1, # the number of past steps to draw 'discrete': False, # whether to use discrete control space 'discrete_acc': [-3.0, 0.0, 3.0], # discrete value of accelerations 'discrete_steer': [-0.2, 0.0,
```

```

0.2], # discrete value of steering angles 'continuous_accel_range': [-3.0, 3.0], # continuous
acceleration range 'continuous_steer_range': [-0.3, 0.3], # continuous steering angle range
'max_episode_step': scenario_config['max_episode_step'], # maximum timesteps per episode
'max_waypt': 12, # maximum number of waypoints 'lidar_bin': 0.125, # bin size of lidar sensor
(meter) 'out_lane_thres': 4, # threshold for out of lane (meter) 'desired_speed': 8, # desired
speed (m/s) 'image_sz': 1024, # TODO: move to config of od scenario

    # pass config from scenario to agent agent_config['mode'] = scenario_config['mode'] agent_config['eg'
= scenario_config['ego_action_dim'] agent_config['ego_state_dim'] = scenario_config['ego_state_dim']
agent_config['ego_action_limit'] = scenario_config['ego_action_limit']

    # define logger logger_kw_args = setup_logger_kw_args( self.exp_name, self.output_dir,
self.seed, agent=agent_config['policy_type'], scenario=scenario_config['policy_type'], scenario_category=''
) self.logger = Logger(**logger_kw_args)

    # prepare parameters if self.mode == 'train_agent': self.buffer_capacity = agent_config['buffer_capacit
self.eval_in_train_freq = agent_config['eval_in_train_freq'] self.save_freq = agent_config['save_freq']
self.train_episode = agent_config['train_episode'] self.current_episode = -1 self.logger.save_config(agent_c
self.logger.create_training_dir() elif self.mode == 'train_scenario': self.save_freq = agent_config['save_freq
self.logger.create_eval_dir(load_existing_results=False) elif self.mode == 'eval': self.save_freq
= agent_config['save_freq'] self.logger.log('» Evaluation Mode, skip config saving', 'yellow')
self.logger.create_eval_dir(load_existing_results=False) else: raise NotImplementedError(f'Unsupported
mode: {self.mode}."))

    # define agent and scenario self.logger.log('» Agent Policy: ' + agent_config['policy_type'])
self.logger.log('» Scenario Policy: ' + scenario_config['policy_type'])

    if self.scenario_config['auto_ego']: self.logger.log('» Using auto-polit for ego vehicle, ac
tion of policy will be ignored', 'yellow') if scenario_config['policy_type'] == 'ordinary' and
self.mode != 'train_agent': self.logger.log('» Ordinary scenario can only be used in agent trainin
ing', 'red') raise Exception() self.logger.log('» ' + '-' * 40)

    # define agent and scenario policy self.agent_policy = AGENT_POLICY_LIST[agent_config['policy_'
logger=self.logger) self.scenario_policy = SCENARIO_POLICY_LIST[scenario_config['policy_type']][(sc
logger=self.logger) if self.save_video: assert self.mode == 'eval', "only allow video saving in
eval mode" self.logger.init_video_recorder()

    def_init_world(self): self.logger.log("» Initializing carla world") self.world = self.client.get_world()
settings = self.world.get_settings() settings.synchronous_mode = True settings.fixed_delta_seconds
= self.fixed_delta_seconds self.world.apply_settings(settings) CarlaDataProvider.set_client(self.client)
CarlaDataProvider.set_world(self.world) CarlaDataProvider.set_traffic_manager_port(self.scenario_config

    def_init_scenic(self, config): self.logger.log(f"» Initializing scenic simulator: {config.scenic_file}")
self.scenic = ScenicSimulator(config.scenic_file, config.extra_params)

    def_init_renderer(self): self.logger.log("» Initializing pygame birdeye renderer") pygame.init()

```

```

flag = pygame.HWSURFACE | pygame.DOUBLEBUF if not self.render: flag = flag | pygame.HIDDEN
if self.scenario_category in ['planning', 'scenic']: # [bird-eye view, Lidar, front view] or [bird-
eye view, front view] if self.env_params['disable_lidar']: window_size = (self.env_params['display_size'] *
* 2, self.env_params['display_size'] * self.num_scenario) else: window_size = (self.env_params['display_size'] *
* 3, self.env_params['display_size'] * self.num_scenario) else: window_size = (self.env_params['display_size'],
self.env_params['display_size'] * self.num_scenario) self.display = pygame.display.set_mode(window_size, flag)

# initialize the render for generating observation and visualization pixels_per_meter =
self.env_params['display_size'] / self.env_params['obs_range'] pixels_ahead_vehicle = (self.env_params['
/ 2 - self.env_params['d_behind']] * pixels_per_meter self.birdeye_params = 'screen_size':
[self.env_params['display_size'], self.env_params['display_size']], 'pixels_per_meter': pix-
els_per_meter, 'pixels_ahead_vehicle': pixels_ahead_vehicle, self.birdeye_render = Birdey-
eRender(self.world, self.birdeye_params, logger=self.logger)

def run_scenes(self, scenes): self.logger.log(f'» Begin to run the scene...") ## currently
there is only one scene in this list ## for scene in scenes: if self.scenic.setSimulation(scene):
self.scenic.update_behavior = self.scenic.runSimulation() next(self.scenic.update_behavior)

def train(self, data_loader, start_episode=0, replay_buffer = None): # general buffer for
both agent and scenario
    for _ in tqdm(range(len(data_loader))): self.current_episode += 1 if self.current_episode
>= self.train_episode: return if self.current_episode < start_episode: continue # sample sce-
narios sampled_scenario_configs, _ = data_loader.sampler() # reset the index counter to create
endless loader # data_loader.reset_idx_counter()

    scenes = [config.scene for config in sampled_scenario_configs] # begin to run the scene
    self.run_scenes(scenes)

    # get static obs and then reset with init action static_obs = self.env.get_static_obs(sampled_scenario_
    self.scenario_policy.load_model(sampled_scenario_configs) scenario_init_action, additional_dict
    = self.scenario_policy.get_init_action(static_obs) obs, infos = self.env.reset(sampled_scenario_configs,
    scenario_init_action) replay_buffer.store_init([static_obs, scenario_init_action], additional_dict=additional_
    # get ego vehicle from scenario self.agent_policy.set_ego_and_route(self.env.get_ego_vehicles(),
    infos)

    # start loop episode_reward = [] while not self.env.all_scenario_done(): # get action from
    agent policy and scenario policy (assume using one batch) ego_actions = self.agent_policy.get_action(obs,
    infos, deterministic=False) scenario_actions = self.scenario_policy.get_action(obs, infos, deter-
    ministic=False)

    # apply action to env and get obs next_obs, rewards, dones, infos = self.env.step(ego_actions=ego_action_
    scenario_actions=scenario_actions) replay_buffer.store([ego_actions, scenario_actions, obs, next_obs,
    rewards, dones], additional_dict=infos) obs = copy.deepcopy(next_obs) episode_reward.append(np.mean(re-
    wards))

```

```
# train off-policy agent or scenario if self.mode == 'train_agent' and self.agent_policy.type
== 'offpolicy': loss = self.agent_policy.train(replay_buffer) elif self.mode == 'train_scenario'
and self.scenario_policy.type == 'offpolicy': self.scenario_policy.train(replay_buffer)

score_function = get_route_scores if self.scenario_category in ['planning', 'scenic'] else
get_perception_scores all_scores = score_function(self.env.running_results)

# end up environment self.env.clean_up() replay_buffer.finish_one_episode() self.logger.add_training_
self.current_episode) self.logger.add_training_results('episode_reward', np.sum(episode_reward))
for key, value in all_scores.items(): self.logger.add_training_results(key, value) if loss is not
None: critic_loss, actor_loss = loss self.logger.add_training_results('critic_loss', critic_loss)
self.logger.add_training_results('actor_loss', actor_loss) else: critic_loss, actor_loss = 0, 0 self.logger.add_t_
critic_loss) self.logger.add_training_results('actor_loss', actor_loss) self.logger.log(f'Episode:
self.current_episode, #buffer_len: replay_buffer.buffer_len, critic: critic_loss:.3f, actor: ac-
tor_loss:.3f') self.logger.save_training_results()

# train on-policy agent or scenario if self.mode == 'train_agent' and self.agent_policy.type
== 'onpolicy': self.agent_policy.train(replay_buffer) elif self.mode == 'train_scenario' and self.scenario_p
in ['init_state', 'onpolicy']: self.scenario_policy.train(replay_buffer)

# eval during training if (self.current_episode+1) % self.eval_in_train_freq == 0: #self.eval(env,
data_loader) pass

# save checkpoints if (self.current_episode+1) % self.save_freq == 0: if self.mode ==
'train_agent': self.agent_policy.save_model(self.current_episode, replay_buffer) if self.mode
== 'train_scenario': self.scenario_policy.save_model(self.current_episode)

self.scenic.destroy()

def eval(self, data_loader, select=False): num_finished_scenario = 0 data_loader.reset_idx_counter()
# recording the score and the id of corresponding selected scenes map_id_score = behav-
ior_name = data_loader.behavior route_id = data_loader.route_id opt_step = data_loader.opt_step
opt_time = 0

if route_id is None: log_name = f'OPT_behavior_name' else: log_name = f'OPT_behavior_name_RO
route_id'

if select: self.scene_map[log_name] = self.scene_map[log_name][f'opt_time_opt_time']
= self.scenic.save_params()

while len(data_loader) > 0: # sample scenarios sampled_scenario_configs, num_sampled_scenario
= data_loader.sampler() num_finished_scenario += num_sampled_scenario assert num_sampled_scenario
== 1, 'scenic can only run one scene at one time'

scenes = [config.scene for config in sampled_scenario_configs] # begin to run the scene
self.run_scenes(scenes)

# reset envs with new config, get init action from scenario policy, and run scenario static_obs
= self.env.get_static_obs(sampled_scenario_configs) self.scenario_policy.load_model(sampled_scenario_c
```

```

scenario_init_action, _ = self.scenario_policy.get_init_action(static_obs, deterministic=True)
obs, infos = self.env.reset(sampled_scenario_configs, scenario_init_action)

    # get ego vehicle from scenario self.agent_policy.set_ego_and_route(self.env.get_ego_vehicles(),
    infos)

        score_list = s_i: [] for s_i in range(num_sampled_scenario) while not self.env.all_scenario_done():

# get action from agent policy and scenario policy (assume using one batch) ego_actions =
self.agent_policy.get_action(obs, infos, deterministic=True) scenario_actions = self.scenario_policy.get_act
infos, deterministic=True)

        # apply action to env and get obs obs, rewards, _, infos = self.env.step(ego_actions=ego_actions,
scenario_actions=scenario_actions)

        # save video if self.save_video: if self.scenario_category in ['planning', 'scenic']: self.logger.add_fram
0, 2)) else: self.logger.add_frame(s_i['scenario_id']): ego_actions[n_i]['annotated_image'] for
n_i, s_i in enumerate(infos))

        # accumulate scores of corresponding scenario reward_idx = 0 for s_i in infos: score = re-
wards[reward_idx] if self.scenario_category in ['planning', 'scenic'] else 1-infos[reward_idx]['iou_loss']
score_list[s_i['scenario_id']].append(score) reward_idx += 1

        # clean up all things self.logger.log("» All scenarios are completed. Clearning up all ac-
tors") self.env.clean_up()

        # save video if self.save_video: data_ids = [config.data_id for config in sampled_scenario_configs]
self.logger.save_video(data_ids=data_ids, log_name=log_name)

        # print score for ranking self.logger.log(f'[num_finished_scenario/{data_loader.num_total_scenar
Ranking scores for batch scenario:', color='yellow') for s_i in score_list.keys(): self.logger.log('Env
id ' + str(s_i) + ': ' + str(np.mean(score_list[s_i])), color='yellow')

        # calculate evaluation results score_function = get_route_scores if self.scenario_category
in ['planning', 'scenic'] else get_perception_scores all_running_results = self.logger.add_eval_results(recor
all_scores = score_function(all_running_results) self.logger.add_eval_results(scores=all_scores)
self.logger.print_eval_results() if len(self.env.running_results) % self.save_freq == 0: self.logger.save_eval_
if infos[0]['collision']: self.scenic.record_params() if select and (num_finished_scenar
% opt_step == 0): opt_time += 1 self.scenic.update_params() self.scene_map[log_name][f'opt_time_{opt_time}'] =
self.scenic.save_params() data_loader.train_scene(opt_time)

        self.logger.save_eval_results(log_name)

        if select: self.scene_map[log_name]['select_id'] = self.select_adv_scene(self.logger.eval_records,
score_function, data_loader.select_num) self.dump_scene_map(sampled_scenario_configs[0].scenario_id)

        self.logger.clear() self.scenic.destroy()

    def select_adv_scene(self, results, score_function, select_num): # define your own se-
lection mechanism here map_id_score_collision = map_id_score_non_collision = for i in re-
sults.keys(): score = score_function(i:results[i]) if score['collision_rate'] == 1: map_id_score_collision[i]

```

```

= score['final_score'] else: map_id_score_non_collision[i] = score['final_score']

# Sort the collision scenes by their scores collision_scenes_sorted = sorted(map_id_score_collision.items(),
key=lambda x: x[1])

# Get the number of scenes to select from the collision cases num_collision_selected =
min(select_num, len(collision_scenes_sorted))

# Select the lowest scored scenes with collision selected_scene_id = [scene[0] for scene in
collision_scenes_sorted[:num_collision_selected]]

# If not enough collision scenes, select remaining scenes num_non_collision_selected = se-
lect_num - num_collision_selected if num_non_collision_selected > 0: # Sort the non-collision
scenes by their scores non_collision_scenes_sorted = sorted(map_id_score_non_collision.items(),
key=lambda x: x[1]) # Select the lowest scored scenes from the non-collision cases selected_scene_id.extend(
for scene in non_collision_scenes_sorted[:num_non_collision_selected]]) return sorted(selected_scene_id)

def run(self, test_epoch=None): # get scenario data of different maps config_list = scenic_parse(self.scenics,
self.logger)

    ### load rl model ## if self.mode == 'train_scenario': ## we only need the pretrained
    surrogate model here ## pass elif self.mode == 'train_agent': ## initlize buffer ### Buffer =
    RouteReplayBuffer if self.scenario_category in ['scenic', 'planning'] else PerceptionReplay-
    Buffer replay_buffer = Buffer(self.num_scenario, self.mode, self.buffer_capacity)

    ### repeat the training, 20 is just a random placeholder config_list = config_list * 20

    ### check if resume ### if self.continue_agent_training: self.logger.load_training_results()
    start_episode = self.check_continue_training(self.agent_policy, replay_buffer) + 1 if start_episode
    >= self.train_episode: return else: self.clean_cache(self.agent_policy.model_path) start_episode
    = -1

    elif self.mode == 'eval': ### load trained model for evaluation ### if test_epoch: self.agent_policy.load(
    last_town = None for config in config_list:
        ## set log name ## if config.route_id is None: log_name = f'OPT_config.behavior' else:
        log_name = f'OPT_config.behavior_ROUTE-config.route_id'

        ## check if all done ## if self.mode == 'eval': if self.logger.check_eval_dir(log_name) ==
        config.select_num: self.logger.log(f'» This scenario and route have been done.') continue elif
        self.mode == 'train_agent': if self.current_episode >= self.train_episode - 1: return

        if self.current_episode + config.select_num < start_episode: self.current_episode += con-
        fig.select_num continue

        # initialize scenic self._init_scenic(config)
        # initialize map and render if last_town != config.extra_params['town']: self._init_world()
        self._init_renderer() last_town = config.extra_params['town'] self.world.scenic = self.scenic

        # create scenarios within the vectorized wrapper self.env = VectorWrapper( self.env_params,
        self.scenario_config, self.world, self.birdeye_render, self.display, self.logger )

```

```

# prepare data loader and buffer
data_loader = ScenicDataLoader(self.scenic, config, self.num_scenarios)
# run with different modes
if self.mode == 'train_scenario': ### select hard scenic scenario config on the surrogate
model ### self.scene_map = self.load_scene_map(config.scenario_id) self.agent_policy.set_mode('eval')
self.scenario_policy.set_mode('eval') self.eval(data_loader, select = True) elif self.mode ==
'train_agent': ### train the surrogate model on the selected hard scenarios ### self.agent_policy.set_mode('tr
self.scenario_policy.set_mode('eval') self.train(data_loader, start_episode, replay_buffer) elif
self.mode == 'eval': ### evaluate the trained agent on different test models ### self.agent_policy.set_mode(
self.scenario_policy.set_mode('eval') self.eval(data_loader) else: raise NotImplementedError(f'Unsupported
mode: {self.mode}.')
def check_continue_training(self, policy, replay_buffer): # load previous checkpoint
policy.load_model(replay_buffer=replay_buffer) if policy.continue_episode == 0: start_episode =
0 self.logger.log('» Previous checkpoint not found. Training from scratch.') else: start_episode =
policy.continue_episode self.logger.log(f'» Continue training from previous checkpoint, epoch:
start_episode.') return start_episode
def dump_scene_map(self, scenario_id): # load previous checkpoint
scenic_dir = os.path.join(self.scen
f'scenario_{scenario_id}') f = open(os.path.join(scenic_dir, f'{scenic_dir.split("/")[-1]}.json'), 'w')
json.dumps_str = json.dumps(self.scene_map, indent=4) print(json.dumps_str, file=f) f.close()
def load_scene_map(self, scenario_id): # load previous checkpoint
scenic_dir = os.path.join(self.scen
f'scenario_{scenario_id}') try: with open(os.path.join(scenic_dir, f'{scenic_dir.split("/")[-1]}.json'),
'r') as f: data = json.loads(f.read()) except: data = None
def clean_cache(self, path): # Get a list of all files in directory
all_files = glob.glob(os.path.join(path,
'*'))
# Specify the file to keep
file_to_keep = os.path.join(path, 'model.sac.-001.torch')
# Remove all files except the one to keep
for file in all_files:
    if file != file_to_keep:
        os.remove(file)
def close(self):
    pygame.quit() # close pygame renderer
    if self.env: self.env.clean_up()

```

A.4 scene_runner

''' Date: 2023-01-31 22:23:17 LastEditTime: 2023-03-07 12:28:17 Description: Copyright (c) 2022-2023 Safebench Team

This work is licensed under the terms of the MIT license. For a copy, see <<https://opensource.org/licenses/MIT>>'''

```

import copy import os import json import glob import random
import numpy as np import carla import pygame from tqdm import tqdm
from safebench.gym_carla.env_wrapper import VectorWrapper from safebench.gym_carla.envs.render
import BirdeyeRender from safebench.gym_carla.replay_buffer import RouteReplayBuffer, Per-

```

ceptionReplayBuffer

```

from safebench.agent import AGENT_POLICY_LIST from safebench.scenario import SCE-
NARIO_POLICY_LIST

from safebench.scenario.scenario_manager.carla_data_provider import CarlaDataProvider
from safebench.scenario.scenario_data_loader import ScenarioDataLoader, ScenicDataLoader
from safebench.scenario.tools.scenario_utils import scenario_parse, dynamic_scenic_parse

from safebench.util.logger import Logger, setup_logger_kwarg from safebench.util.metric_util
import get_route_scores, get_perception_scores from safebench.util.scenic_utils import Scen-
icSimulator

class ScenicRunner: def __init__(self, agent_config, scenario_config): self.scenario_config
= scenario_config self.agent_config = agent_config

    self.seed = scenario_config['seed'] self.exp_name = scenario_config['exp_name'] self.output_dir
= scenario_config['output_dir'] self.mode = scenario_config['mode'] self.save_video = sce-
nario_config['save_video']

    self.render = scenario_config['render'] self.num_scenario = scenario_config['num_scenario']
self.fixed_delta_seconds = scenario_config['fixed_delta_seconds'] self.scenario_category = sce-
nario_config['scenario_category']

    # continue training flag self.continue_agent_training = scenario_config['continue_agent_training']
self.continue_scenario_training = scenario_config['continue_scenario_training']

    # apply settings to carla self.client = carla.Client('localhost', scenario_config['port']) self.client.set_time
self.world = None self.env = None

    self.env_params = 'auto_ego': scenario_config['auto_ego'], 'obs_type': agent_config['obs_type'],
'scenario_category': self.scenario_category, 'ROOT_DIR': scenario_config['ROOT_DIR'], 'warm_up_st
9, # number of ticks after spawning the vehicles 'disable_lidar': True, # show bird-eye view li-
dar or not 'display_size': 128, # screen size of one bird-eye view window 'obs_range': 32, # ob-
servation range (meter) 'd_behind': 12, # distance behind the ego vehicle (meter) 'max_past_step':
1, # the number of past steps to draw 'discrete': False, # whether to use discrete control space
'discrete_acc': [-3.0, 0.0, 3.0], # discrete value of accelerations 'discrete_steer': [-0.2, 0.0,
0.2], # discrete value of steering angles 'continuous_accel_range': [-3.0, 3.0], # continuous
acceleration range 'continuous_steer_range': [-0.3, 0.3], # continuous steering angle range
'max_episode_step': scenario_config['max_episode_step'], # maximum timesteps per episode
'max_waypt': 12, # maximum number of waypoints 'lidar_bin': 0.125, # bin size of lidar sen-
sor (meter) 'out_lane_thres': 4, # threshold for out of lane (meter) 'desired_speed': 8, # desired
speed (m/s) 'image_sz': 1024, # TODO: move to config of od scenario

    # pass config from scenario to agent agent_config['mode'] = scenario_config['mode'] agent_config['eg
= scenario_config['ego_action_dim'] agent_config['ego_state_dim'] = scenario_config['ego_state_dim']
agent_config['ego_action_limit'] = scenario_config['ego_action_limit']

```

```
# define logger logger_kw_args = setup_logger_kw_args( self.exp_name, self.output_dir,
self.seed, agent=agent_config['policy_type'], scenario=scenario_config['policy_type'], scenario_category=scenario_config['category'])
) self.logger = Logger(**logger_kw_args)

# prepare parameters if self.mode == 'train_agent': self.buffer_capacity=agent_config['buffer_capacity']
self.eval_in_train_freq=agent_config['eval_in_train_freq'] self.save_freq=agent_config['save_freq']
self.train_episode=agent_config['train_episode'] self.current_episode=-1 self.logger.save_config(agent_config)
self.logger.create_training_dir() elif self.mode == 'train_scenario': self.save_freq=agent_config['save_freq']
self.logger.create_eval_dir(load_existing_results=False) elif self.mode == 'eval': self.save_freq
= agent_config['save_freq'] self.logger.log('» Evaluation Mode, skip config saving', 'yellow')
self.logger.create_eval_dir(load_existing_results=False) else: raise NotImplementedError(f"Unsupported mode: {self.mode}.") )

# define agent and scenario self.logger.log('» Agent Policy: ' + agent_config['policy_type'])
self.logger.log('» Scenario Policy: ' + scenario_config['policy_type'])

if self.scenario_config['auto_ego']: self.logger.log('» Using auto-polit for ego vehicle, ac-
tion of policy will be ignored', 'yellow') if scenario_config['policy_type'] == 'ordinary' and
self.mode != 'train_agent': self.logger.log('» Ordinary scenario can only be used in agent train-
ing', 'red') raise Exception() self.logger.log('» ' + '-' * 40)

# define agent and scenario policy self.agent_policy=AGENT_POLICY_LIST[agent_config['policy_type']]
logger=self.logger) self.scenario_policy=SCENARIO_POLICY_LIST[scenario_config['policy_type']])(so
logger=self.logger) if self.save_video: assert self.mode == 'eval', "only allow video saving in
eval mode" self.logger.init_video_recorder()

def _init_world(self): self.logger.log("» Initializing carla world") self.world = self.client.get_world()
settings = self.world.get_settings() settings.synchronous_mode = True settings.fixed_delta_seconds
= self.fixed_delta_seconds self.world.apply_settings(settings) CarlaDataProvider.set_client(self.client)
CarlaDataProvider.set_world(self.world) CarlaDataProvider.set_traffic_manager_port(self.scenario_config['tm_port'])

def _init_scenic(self, config): self.logger.log(f"» Initializing scenic simulator: {config.scenic_file}")
self.scenic = ScenicSimulator(config.scenic_file, config.extra_params)

def _init_renderer(self): self.logger.log("» Initializing pygame birdeye renderer") pygame.init()
flag=pygame.HWSURFACE|pygame.DOUBLEBUF if not self.render: flag=flag|pygame.HIDDEN
if self.scenario_category in ['planning', 'scenic']: # [bird-eye view, Lidar, front view] or [bird-
eye view, front view] if self.env_params['disable_lidar']: window_size=(self.env_params['display_size']
* 2, self.env_params['display_size'] * self.num_scenario) else: window_size=(self.env_params['display_size']
* 3, self.env_params['display_size'] * self.num_scenario) else: window_size=(self.env_params['display_size']
self.env_params['display_size'] * self.num_scenario) self.display=pygame.display.set_mode(window_size)
flag)

# initialize the render for generating observation and visualization pixels_per_meter =
self.env_params['display_size'] / self.env_params['obs_range'] pixels_ahead_vehicle=(self.env_params['display_size'] /
self.env_params['obs_range']) * pixels_per_meter
```

```

/ 2 - self.env_params['d_behind']) * pixels_per_meter self.birdeye_params = 'screen_size':
[self.env_params['display_size'], self.env_params['display_size']], 'pixels_per_meter': pixels_per_meter, 'pixels_ahead_vehicle': pixels_ahead_vehicle, self.birdeye_render = BirdeyeRender(self.world, self.birdeye_params, logger=self.logger)

    def run_scenes(self, scenes): self.logger.log(f'» Begin to run the scene...") ## currently
there is only one scene in this list ## for scene in scenes: if self.scenic.setSimulation(scene):
self.scenic.update_behavior = self.scenic.runSimulation() next(self.scenic.update_behavior)

    def train(self, data_loader, start_episode=0, replay_buffer = None): # general buffer for
both agent and scenario
        for _ in tqdm(range(len(data_loader))): self.current_episode += 1 if self.current_episode
>= self.train_episode: return if self.current_episode < start_episode: continue # sample sce-
narios sampled_scenario_configs, _ = data_loader.sampler() # reset the index counter to create
endless loader # data_loader.reset_idx_counter()

        scenes = [config.scene for config in sampled_scenario_configs] # begin to run the scene
self.run_scenes(scenes)

        # get static obs and then reset with init action static_obs = self.env.get_static_obs(sampled_scenario_con
self.scenario_policy.load_model(sampled_scenario_configs) scenario_init_action, additional_dict
= self.scenario_policy.get_init_action(static_obs) obs, infos = self.env.reset(sampled_scenario_configs,
scenario_init_action) replay_buffer.store_init([static_obs, scenario_init_action], additional_dict=additional_
dict)

        # get ego vehicle from scenario self.agent_policy.set_ego_and_route(self.env.get_ego_vehicles(),
infos)

        # start loop episode_reward = [] while not self.env.all_scenario_done(): # get action from
agent policy and scenario policy (assume using one batch) ego_actions = self.agent_policy.get_action(obs,
infos, deterministic=False) scenario_actions = self.scenario_policy.get_action(obs, infos, deter-
ministic=False)

        # apply action to env and get obs next_obs, rewards, dones, infos = self.env.step(ego_actions=ego_action
scenario_actions=scenario_actions) replay_buffer.store([ego_actions, scenario_actions, obs, next_obs,
rewards, dones], additional_dict=infos) obs = copy.deepcopy(next_obs) episode_reward.append(np.mean(re
wards))

        # train off-policy agent or scenario if self.mode == 'train_agent' and self.agent_policy.type
== 'offpolicy': loss = self.agent_policy.train(replay_buffer) elif self.mode == 'train_scenario'
and self.scenario_policy.type == 'offpolicy': self.scenario_policy.train(replay_buffer)

        score_function = get_route_scores if self.scenario_category in ['planning', 'scenic'] else
get_perception_scores all_scores = score_function(self.env.running_results)

        # end up environment self.env.clean_up() replay_buffer.finish_one_episode() self.logger.add_training_
self.current_episode) self.logger.add_training_results('episode_reward', np.sum(episode_reward))
for key, value in all_scores.items(): self.logger.add_training_results(key, value) if loss is not
None: critic_loss, actor_loss = loss self.logger.add_training_results('critic_loss', critic_loss)

```

```

self.logger.add_training_results('actor_loss', actor_loss) else: critic_loss, actor_loss = 0, 0 self.logger.add_t
critic_loss) self.logger.add_training_results('actor_loss', actor_loss) self.logger.log(f'Episode:
self.current_episode, #buffer_len: replay_buffer.buffer_len, critic: critic_loss:.3f, actor: ac-
tor_loss:.3f') self.logger.save_training_results()

    # train on-policy agent or scenario if self.mode == 'train_agent' and self.agent_policy.type
    == 'onpolicy': self.agent_policy.train(replay_buffer) elif self.mode == 'train_scenario' and self.scenario_p
in ['init_state', 'onpolicy']: self.scenario_policy.train(replay_buffer)

    # eval during training if (self.current_episode+1) % self.eval_in_train_freq == 0: #self.eval(env,
data_loader) pass

    # save checkpoints if (self.current_episode+1) % self.save_freq == 0: if self.mode ==
'train_agent': self.agent_policy.save_model(self.current_episode, replay_buffer) if self.mode
== 'train_scenario': self.scenario_policy.save_model(self.current_episode)

    self.scenic.destroy()

def eval(self, data_loader, select=False): num_finished_scenario = 0 data_loader.reset_idx_counter()
# recording the score and the id of corresponding selected scenes map_id_score = behav
ior_name = data_loader.behavior opt_step = data_loader.opt_step opt_time = 0
log_name = f'OPT_behavior_{behavior_name}'

if select: self.scene_map[log_name] = self.scene_map[log_name][f'opt_time_opt_time']
= self.scenic.save_params()

while len(data_loader) > 0: # sample scenarios sampled_scenario_configs, num_sampled_scenario
= data_loader.sampler() num_finished_scenario += num_sampled_scenario assert num_sampled_scenario
== 1, 'scenic can only run one scene at one time'

scenes = [config.scene for config in sampled_scenario_configs] # begin to run the scene
self.run_scenes(scenes)

# reset envs with new config, get init action from scenario policy, and run scenario static_obs
= self.env.get_static_obs(sampled_scenario_configs) self.scenario_policy.load_model(sampled_scenario_c
scenario_init_action, _ = self.scenario_policy.get_init_action(static_obs, deterministic=True)
obs, infos = self.env.reset(sampled_scenario_configs, scenario_init_action)

# get ego vehicle from scenario self.agent_policy.set_ego_and_route(self.env.get_ego_vehicles(),
infos)

score_list = s_i: [] for s_i in range(num_sampled_scenario) while not self.env.all_scenario_done():
# get action from agent policy and scenario policy (assume using one batch) ego_actions =
self.agent_policy.get_action(obs, infos, deterministic=True) scenario_actions = self.scenario_policy.get_ac
tions(obs, infos, deterministic=True)

# apply action to env and get obs obs, rewards, _, infos = self.env.step(ego_actions=ego_actions,
scenario_actions=scenario_actions)

# save video if self.save_video: self.logger.add_frame(pygame.surfarray.array3d(self.display).transpos

```

```

0, 2))

# accumulate scores of corresponding scenario reward_idx = 0 for s_i in infos: score = rewards[reward_idx] if self.scenario_category in ['planning', 'scenic'] else 1-infos[reward_idx]['iou_loss']
score_list[s_i['scenario_id']].append(score) reward_idx += 1

# clean up all things self.logger.log("» All scenarios are completed. Clearning up all actors") self.env.clean_up()

# save video if self.save_video: data_ids=[config.data_id for config in sampled_scenario_configs]
self.logger.save_video(data_ids=data_ids, log_name=log_name)

# print score for ranking self.logger.log(f'[num_finished_scenario/{data_loader.num_total_scenario}]')
Ranking scores for batch scenario: ', color='yellow') for s_i in score_list.keys(): self.logger.log('Env id ' + str(s_i) + ': ' + str(np.mean(score_list[s_i])), color='yellow')

# calculate evaluation results score_function = get_route_scores if self.scenario_category in ['planning', 'scenic'] else get_perception_scores
all_running_results = self.logger.add_eval_results(records)
all_scores = score_function(all_running_results) self.logger.add_eval_results(scores=all_scores)
self.logger.print_eval_results() if len(self.env.running_results) % self.save_freq == 0: self.logger.save_eval()

if infos[0]['collision']: self.scenic.record_params()

if select and (num_finished_scenario % opt_step == 0): opt_time += 1 self.scenic.update_params()
self.scene_map[log_name][f'opt_time_opt_time'] = self.scenic.save_params() data_loader.train_scene(opt_time)
self.logger.save_eval_results(log_name)

if select: self.scene_map[log_name]['select_id'] = self.select_adv_scene(self.logger.eval_records,
score_function, data_loader.select_num) self.dump_scene_map()
self.logger.clear() self.scenic.destroy()

def select_adv_scene(self, results, score_function, select_num): # define your own selection mechanism here
map_id_score_collision = map_id_score_non_collision = {}
for i in results.keys(): score = score_function(i:results[i]) if score['collision_rate'] == 1: map_id_score_collision[i] = score['final_score'] else: map_id_score_non_collision[i] = score['final_score']

# Sort the collision scenes by their scores
collision_scenes_sorted = sorted(map_id_score_collision.items(),
key=lambda x: x[1])

# Get the number of scenes to select from the collision cases
num_collision_selected = min(select_num, len(collision_scenes_sorted))

# Select the lowest scored scenes with collision
selected_scene_id = [scene[0] for scene in collision_scenes_sorted[:num_collision_selected]]

# If not enough collision scenes, select remaining scenes
num_non_collision_selected = select_num - num_collision_selected
if num_non_collision_selected > 0: # Sort the non-collision scenes by their scores
non_collision_scenes_sorted = sorted(map_id_score_non_collision.items(),
key=lambda x: x[1])
# Select the lowest scored scenes from the non-collision cases
selected_scene_id.extend([scene[0] for scene in non_collision_scenes_sorted[:num_non_collision_selected]])
return sorted(selected_scene_id)

```

```

def run(self, test_epoch = None): # get scenario data of different maps config_list = dynamic_scenic_parse(self.scenario_config, self.logger)

    ### load rl model ## if self.mode == 'train_scenario': ## we only need the pretrained
    surrogate model here ## pass elif self.mode == 'train_agent': ## initlize buffer ### Buffer =
    RouteReplayBuffer if self.scenario_category in ['scenic', 'planning'] else PerceptionReplay-
    Buffer replay_buffer = Buffer(self.num_scenario, self.mode, self.buffer_capacity)

    ### repeat the training, 20 is just a random placeholder config_list = config_list * 20

    ### check if resume ### if self.continue_agent_training: self.logger.load_training_results()
    start_episode = self.check_continue_training(self.agent_policy, replay_buffer) + 1 if start_episode
    >= self.train_episode: return else: self.clean_cache(self.agent_policy.model_path) start_episode
    = -1

    elif self.mode == 'eval' and test_epoch: ### load trained model for evaluation ### self.agent_policy.loa
    last_town = None for config in config_list:

        ## set log name ## log_name = f'OPT_config.behavior'

        ## check if all done ## if self.mode == 'eval': if self.logger.check_eval_dir(log_name) ==
        config.select_num: self.logger.log(f'» This scenario and route have been done.') continue elif
        self.mode == 'train_agent': if self.current_episode >= self.train_episode - 1: return

        if self.current_episode + config.select_num < start_episode: self.current_episode += config.select_num continue

        # initialize scenic self._init_scenic(config)
        # initialize map and render self._init_world() self._init_renderer() self.world.scenic = self.scenic
        # create scenarios within the vectorized wrapper self.env = VectorWrapper( self.env_params,
        self.scenario_config, self.world, self.birdeye_render, self.display, self.logger )
        # prepare data loader and buffer data_loader = ScenicDataLoader(self.scenic, config, self.num_scenario)
        # run with different modes

        if self.mode == 'train_scenario': ### select hard scenic scenario config on the surrogate
        model ### self.scene_map = self.load_scene_map() self.agent_policy.set_mode('eval') self.scenario_policy
        self.eval(data_loader, select = True) elif self.mode == 'train_agent': ### train the surrogate
        model on the selected hard scenrios ### self.agent_policy.set_mode('train') self.scenario_policy.set_mode('
        self.train(data_loader, start_episode, replay_buffer) elif self.mode == 'eval': ### evaluate the
        trained agent on different test models ### self.agent_policy.set_mode('eval') self.scenario_policy.set_mode('
        self.eval(data_loader) else: raise NotImplementedError(f'Unsupported mode: self.mode.')

        def check_continue_training(self, policy, replay_buffer): # load previous checkpoint pol-
        icy.load_model(replay_buffer = replay_buffer) if policy.continue_episode == 0: start_episode =
        0 self.logger.log('» Previous checkpoint not found. Training from scratch.') else: start_episode
        = policy.continue_episode self.logger.log(f'» Continue training from previous checkpoint, epoch:
        start_episode.') return start_episode

```

```
def dump_scene_map(self): # load previous checkpoint
scenic_dir = self.scenario_config['scenic_dir']
f = open(os.path.join(scenic_dir, f'dynamic_scenario.json'), 'w')
json.dumps_str = json.dumps(self.scene,
indent=4)
print(json.dumps_str, file=f)
f.close()

def load_scene_map(self): # load previous checkpoint
scenic_dir = self.scenario_config['scenic_dir']
try:
with open(os.path.join(scenic_dir, f'dynamic_scenario.json'), 'r') as f:
data = json.loads(f.read())
except:
data = None
return data

def clean_cache(self, path): # Get a list of all files in directory
all_files = glob.glob(os.path.join(path, '*'))
# Specify the file to keep
file_to_keep = os.path.join(path, 'model.sac.-001.torch')
# Remove all files except the one to keep
for file in all_files:
if file != file_to_keep:
os.remove(file)

def close(self):
pygame.quit() # close pygame renderer if self.env:
self.env.clean_up()
%
```