



湖南工商大学

HUNAN UNIVERSITY OF TECHNOLOGY AND BUSINESS

毕业设计

基于预训练大模型的高保真三

题 目 维智能驾驶场景生成系统

学生姓名 郑睿翔

学 号 2123030045

学 院 人工智能与先进计算学院

专业班级 大数据与人工智能

指导教师 王海东

职 称 讲师

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 基于预训练大模型的高保真三维智能驾驶场景生成系统 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： 郑睿翔

日期： 2025年4月28日

湖南工商大学本科毕业设计

版权使用授权书

本毕业设计《基于预训练大模型的高保真三维智能驾驶场景生成系统》是本人在校期间所完成学业的组成部分，是在学校教师的指导下完成的。因此，本人特授权学校可将本毕业设计的全部或部分内容编入有关书籍、数据库保存，可采用复制、印刷、网页制作等方式将设计文本和经过编辑、批注等处理的设计文本提供给读者查阅、参考，可向有关学术部门和国家有关教育主管部门呈送复印件和电子文档。本毕业设计无论做何种处理，必须尊重本人的著作权，署明本人姓名。

设计作者(签字)

时间 年 月 日

指导教师已阅(签字)

时间 年 月 日

摘要

当前行业面临着智能驾驶测试场景构建效率低下这一重要问题，为此本文提出一种融合大语言模型（LLM）与形式化场景描述语言的新型生成框架，此框架目的在于高效地把自然语言描述转化成高保真三维测试场景，其包含三大核心模块，第一个是领域知识增强的指令解析模块，该模块能够精准理解自然语言描述里的测试场景需求从而为后续生成工作奠定基础，第二个是语法 - 语义双验证代码生成机制，此机制可确保生成的 Scenic 代码不仅语法正确而且符合实际场景的语义逻辑进而提高代码的可用性和准确性，第三个是物理规则驱动的场景合成引擎，该引擎依据物理规则合理生成三维场景以此保证场景的真实性和可靠性，在 CARLA 仿真平台的实验验证中该系统展现出优异性能。

关键词： 智能驾驶 测试场景 大语言模型 场景描述语言 Scenic

ABSTRACT

The current industry is confronted with the significant issue of low efficiency in constructing test scenarios for autonomous driving. To address this, this paper proposes a novel generation framework that integrates large language models (LLMs) with formal scenario description languages. The purpose of this framework is to efficiently transform natural language descriptions into high-fidelity 3D test scenarios. It comprises three core modules. The first is a domain knowledge - enhanced instruction parsing module, which can accurately understand the test scenario requirements in natural language descriptions and thus lay the foundation for subsequent generation tasks. The second is a syntax - semantics dual - verification code generation mechanism, which can ensure that the generated Scenic code is not only syntactically correct but also semantically logical in accordance with the actual scenario, thereby enhancing the usability and accuracy of the code. The third is a physics - rule - driven scenario synthesis engine, which can reasonably generate 3D scenarios based on physical rules to ensure the authenticity and reliability of the scenarios. The system has demonstrated excellent performance in experiments conducted on the CARLA simulation platform.

Key words: Intelligent Driving Test Scenarios Large Language Models
Scenario Description Language Scenic

目录

第 1 章 绪论	1
1.1 研究背景	1
1.1.1 智能驾驶测试对海量长尾场景的需求矛盾	1
1.1.2 传统场景构建方法的人力成本与效率瓶颈	2
1.1.3 大语言模型在代码生成领域的突破性进展	2
1.2 国内外研究现状	3
1.2.1 基于大语言模型能够根据人的自然语言指令生成 Scenic 场景代码 .	3
1.2.2 将场景代码合成合理的智能驾驶场景	3
1.2.3 对生成的交通场景进行展示和效果的量化衡量	3
1.3 研究内容	4
1.3.1 面向场景描述的领域知识图谱构建	4
1.3.2 基于 LLM 的语义约束代码生成方法	5
1.3.3 场景物理合理性的多模态验证机制	5
1.4 本文研究结构	5
第 2 章 相关工作	8
2.1 预训练语言模型在场景生成系统中的应用机制	8
2.1.1 基于 Sentence-T5-large 的语义编码机制在本系统中的应用	8
2.1.2 自然语言生成结构化代码的建模逻辑	9
2.2 自然语言到交通场景的语义建模	9
2.2.1 时序关系与交通意图建模	10
2.3 Scenic 语言的形式化语义与表达能力	10
2.3.1 Scenic 的语法设计与概率建模	11
2.4 智能驾驶仿真平台 CARLA 与本系统的集成机制	11
2.4.1 本系统中的 CARLA 场景接入方式	11
2.4.2 Scenic 与 CARLA 的联合仿真执行流程	12
第 3 章 系统需求分析与总体架构设计	13
3.1 系统功能需求分析	13
3.1.1 高保真的驾驶测试需求	13
3.1.2 自然语言驱动的场景生成优势与实现	13
3.2 系统总体架构设计	14
3.2.1 自然语言输入模块	14
3.2.2 结构生成模块	15

3.2.3 carla 仿真模块	17
3.2.4 结果评估分析模块	17
第 4 章 系统设计与实现	19
4.1 开发环境	19
4.1.1 硬件环境	19
4.1.2 软件环境	19
4.2 系统核心功能实现	20
4.2.1 自然语言理解功能	20
4.2.2 场景合成与仿真模块	21
4.2.3 场景评估与展示模块	23
4.3 系统编码与实现	25
4.3.1 系统整体架构与主程序设计	25
4.3.2 主控脚本功能与运行流程	26
4.3.3 动态场景仿真控制与异常处理	26
第 5 章 系统案例测试分析	28
5.1 场景案例测试	28
5.1.1 场景一：自车在道路上行驶时，一名行人突然从左侧前方穿出	28
5.1.2 场景二：两辆自车在直路上前后行驶	29
5.1.3 场景三：一名行人横穿马路而自车在直路上行驶	29
5.1.4 场景四：夜晚行人在人行道上走	29
5.1.5 场景五：自车通过十字路口	30
5.2 评估测试分析	30
5.2.1 准确率分析	32
5.2.2 效率分析	33
5.3 本章小节	34
第 6 章 总结与展望	36
6.1 工作总结	36
6.2 研究不足	36
6.3 后续优化方向	37
参考文献	38
附录 A 附录：系统模块源代码	41
A.1 主要代码	41
致谢	55

第1章 绪论

1.1 研究背景

1.1.1 智能驾驶测试对海量长尾场景的需求矛盾

随着智能驾驶技术快速发展起来，它的测试需求也变得日益复杂且多样化，智能驾驶系统需要在各类复杂多变交通场景下^[1]，展现出可靠性能来确保驾驶安全与舒适，然而现实交通环境里存在海量长尾场景，这些场景虽出现频率低但发生时会对系统构成严峻挑战。

长尾场景的复杂性主要体现在如下几个方面，首先交通环境动态性极强，车辆、行人以及非机动车等交通参与者行为模式千变万化，比如在城市道路当中行人可能突然横穿马路，非机动车可能随意变道或者逆行，^[2]这些行为都可能引发潜在的碰撞风险，智能驾驶系统必须能够准确感知并预测这些行为，然后做出合理决策。其次道路条件的多样性增加了测试场景的复杂性^[3]，从城市快速路到乡村小道，再从高速公路到山区道路，不同道路几何形状、路面状况、交通标志和信号灯等都不一样，智能驾驶系统需要在这些不同道路条件下都能正常运行^[4]。此外天气条件和光照条件的变化，也会对智能驾驶系统的感知和决策造成影响，雨、雪、雾等恶劣天气会降低传感器的性能，不同光照条件像强光、弱光、逆光等会影响视觉系统识别效果。

为了保证智能驾驶系统具备安全性和可靠性，测试过程得覆盖尽可能多的长尾场景，然而这面临着巨大挑战，一方面长尾场景数量庞大几乎无法穷尽，要完全覆盖所有可能场景几乎是不可能任务，另一方面这些场景出现频率较低难以在实际道路测试中频繁遇到，所以传统测试方法往往难以满足智能驾驶系统对海量长尾场景的测试需求，使得测试的充分性和有效性受到限制^[5]。



图 1-1 carla 场景样图

1.1.2 传统场景构建方法的人力成本与效率瓶颈

在智能驾驶测试这个领域当中，传统的场景构建方法主要靠人工设计和开发^[6]，这些方法虽说在一定程度上能够满足测试需求，不过也存在着比较显著的局限性，人工设计场景需要投入大量专业人员，这些人员要具备交通工程、计算机科学和智能驾驶技术等多学科深厚知识，能准确理解并模拟各种复杂交通场景，这样的专业人才相对比较稀缺，培养成本也十分高昂，随着测试需求不断增加，对专业人员的需求也持续增长，这进一步加剧了人力成本方面的压力。

其次人工设计场景的效率相对比较低，设计一个复杂交通场景需经多步骤，像需求分析、场景建模、代码编写和调试等，每一个步骤都要耗费大量时间精力，例如在场景建模阶段要精确去定义道路几何形状、交通参与者初始位置和行为模式等，在代码编写阶段需把这些模型转化成可执行代码，这不仅要求具备专业编程技能，还需反复调试来确保代码正确性和稳定性，所以人工设计场景速度远不能满足智能驾驶系统快速迭代和测试的需求，此外人工设计场景的准确性和一致性难保证，因为不同设计人员理解和经验存在差异，可能致使设计出的场景出现一定的差异，并且在复杂场景中人工设计易遗漏关键细节，进而影响测试结果准确性和可靠性。

1.1.3 大语言模型在代码生成领域的突破性进展

近年来大语言模型在自然语言处理领域取得巨大突破且逐渐拓展到代码生成等领域，大语言模型通过在海量文本数据上预训练学习到语言语法语义和逻辑结构能生成自然流畅且符合逻辑的文本内容，这种能力为解决智能驾驶测试场景构建难题提供新的思路。在代码生成领域大语言模型已经展现出强大能力^[7]，通过对代码数据学习大语言模型能理解代码结构和逻辑生成符合语法规范的代码片段，例如一些基于大语言模型的代码生成工具可根据用户输入自然语言描述自动生成相应代码实现，这些工具在软件开发领域得到广泛应用显著提高代码开发效率和质量。大语言模型在代码生成中的优势主要体现在以下几个方面：

首先它能够快速生成代码大大缩短开发周期，传统人工编写代码需经过需求分析设计编码和调试等多个阶段且每个阶段都耗费大量时间，而大语言模型可根据输入描述直接生成代码减少中间环节提高开发效率^[8]，其次大语言模型生成的代码质量较高，它能学习到代码最佳实践和规范生成的代码不仅符合语法规范还具有良好可读性和可维护性，

此外大语言模型还能根据不同需求生成多样化代码实现为开发者提供更多选择。把大语言模型运用到智能驾驶测试场景构建当中^[9]，能够充分发挥它在代码生成领域的优势来解决传统方法面临的困境，借助将自然语言描述的测试场景需求转变为代码实现，大语言模型能够快速生成高保真的测试场景进而提高场景构建的效率和质量，结合智能驾驶领域的专业知识还可以进一步优化大语言模型的性能让它更好适应智能驾驶测试场景构建的需求。

1.2 国内外研究现状

1.2.1 基于大语言模型能够根据人的自然语言指令生成 Scenic 场景代码

随着大语言模型像 GPT -3、T5 等的广泛应用起来，基于自然语言描述生成仿真场景代码成为自动驾驶仿真研究重要方向，自然语言描述能够以简单且直观的方式传递场景信息^[10]，传统手工编写场景代码的方式存在效率低且灵活性差的问题，所以如何利用自然语言生成交通仿真场景脚本成为该领域核心问题。

大语言模型像 GPT - 3 和 T5 这类^[11]，已经成功应用于自然语言到场景代码的转换工作，经过训练之后，这些模型能够理解复杂的自然语言输入内容，依据相关描述，它们可以生成结构化的 Scenic 脚本文件^[12]，这些模型通过解析用户给出的自然语言指令，来生成符合自动驾驶仿真要求的场景代码，比如，提出一种基于 GPT 的框架，能根据自然语言描述生成含车辆行人交通灯等元素的完整交通场景配置。

在挑战与进展方面大语言模型生成场景代码有一定进展但依旧面临挑战，首先处理自然语言里的歧义和模糊性属于一个难题，复杂场景描述下生成的场景可能无法完全符合预期，其次现有模型生成能力在处理复杂或特殊交通场景时存在局限性，所以增强语言模型语义理解和场景生成准确性仍是活跃研究领域。

1.2.2 将场景代码合成合理的智能驾驶场景

把生成的 Scenic 场景代码转化成合理的智能驾驶仿真场景^[13]，这是实现自动驾驶测试与验证的关键步骤，生成的场景不仅要符合交通方面的规则，还得能够模拟真实的驾驶环境来进行有效测试。场景代码转化到仿真环境方面^[14]，当前的研究重点主要集中在怎样把大语言模型所生成的 Scenic 脚本转化成仿真平台能够执行的场景配置内容。例如，开发出基于 Scenic 的编译器用于把自然语言生成场景描述转化为 CARLA 仿真平台所需配置文件，通过这种方式生成的场景能够在高保真的仿真环境里进行验证。

智能场景合成跟动态行为建模这方面，智能驾驶场景不只是包含像道路、交通标志这类静态元素，还涵盖如车辆行驶、变道、停车等动态行为，研究者们提出了多种动态行为建模的方法，让生成的交通场景能更真实地反映驾驶行为与交通流情况^[15]，例如，提出一种基于行为模型的动态场景合成方法来通过模拟交通参与者行为生成交互式智能驾驶环境。

复杂交通场景面临的挑战是，虽说场景合成方法在持续发展，但生成复杂真实且符合交通规则的场景依旧是技术难题，现有的场景生成方法在应对复杂交通场景时，可能会出现车辆行为不自然以及交通规则不严谨等状况，所以提高场景合成的灵活性和复杂性仍是研究重点。

1.2.3 对生成的交通场景进行展示和效果的量化衡量

在生成交通场景之后对其进行展示以及量化评估是保证场景质量和仿真结果准确性的关键步骤^[16]，通过对场景开展可视化以及量化评估研究人员能够验证场景的有效

性并为后续自动驾驶算法优化提供数据支撑。

可视化展示就是把生成的交通场景借助可视化工具进行展示，从而方便进行人工验证和审阅。我这边提出一种基于关键帧截取的可视化方法^[17]，通过在仿真过程当中截取关键帧图像，以此帮助研究人员快速了解仿真过程里的交通场景，结合深度学习相关技术，自动驾驶系统还可以从这些图像当中提取重要信息，进而对场景开展进一步分析与优化。

提出量化评估指标的情况是，为了能系统地对生成场景的质量进行评估，许多研究都提出了量化评估框架。例如，提出一种涵盖场景语义保真度等多方面的多维度评估方法^[18]，这些评估指标既能反映生成场景的真实性又能助研究人员评估仿真结果有效性。

安全性与性能评估：自动驾驶系统在仿真环境中的表现也需要量化评估。提出一种基于自动驾驶系统安全性的评估框架，通过对车辆在生成场景里碰撞率与通过率等指标进行分析，以此评估自动驾驶系统于不同场景之下的安全性和稳定性，借助这样的量化评估手段，研究人员可识别出潜在的风险和问题并进一步优化自动驾驶算法。

1.3 研究内容

1.3.1 面向场景描述的领域知识图谱构建

在自动驾驶测试场景生成这个事情当中，构建面向场景描述的领域知识图谱是实现高效且准确场景生成的基础，领域知识图谱通过整合自动驾驶领域的专业知识^[19]，像交通规则、道路类型、车辆行为模式以及传感器特性等内容，为自然语言描述的解析和形式化代码的生成提供丰富上下文信息和语义支持。

领域知识图谱构建包含多个关键步骤^[20]，首先要对自动驾驶领域知识做系统梳理与分类，明确知识的层次结构和关联关系，这涵盖对交通场景里实体（像车辆、行人、道路、交通标志等）及其属性（例如位置、速度、类型等）的定义，还有这些实体之间关系（比如车辆与道路的交互、车辆与行人的避让等），构建知识图谱可将这些复杂知识结构化表示出来，方便后续进行查询和推理。

在知识图谱的构建过程当中需要考虑知识动态更新与扩展^[21]，自动驾驶技术处于不断发展中新交通规则车辆类型等不断涌现，所以知识图谱得具备良好可扩展性以及可更新性，通过持续开展知识更新能确保知识图谱始终保持最新状态，进而为场景生成提供准确无误的知识方面支持。

除此之外领域知识图谱的构建还得考虑知识表达与存储方式，知识图谱一般是以图的形式进行存储的^[22]，其中节点所表示的是实体而边表示实体间的关系，这种结构化的存储方式不仅方便知识的查询和推理，还能够支持复杂的知识融合与关联分析，借助知识图谱的构建能够实现对自动驾驶场景描述的深度理解和语义解析，为后续的代码生成以及场景合成提供坚实可靠的基础。

1.3.2 基于 LLM 的语义约束代码生成方法

基于大型语言模型（LLM）的语义约束代码生成方法是达成自动驾驶测试场景高效生成的一项关键技术，大型语言模型（LLM）具备强大的语言理解和生成能力可依据自然语言描述生成高质量形式化代码，不过为保证生成代码的准确性与可靠性需在代码生成过程中引入语义约束机制^[23]。

语义约束代码生成方法的关键是把自然语言描述的语义信息转成代码生成约束条件，这些约束条件涵盖交通规则、道路类型以及车辆行为模式等内容，目的是保证生成的代码既符合语法规范又能满足实际场景语义要求^[24]，借助语义约束机制可有效避免生成代码里的逻辑错误与不符合实际场景的状况，以此提高代码的质量和可用性。

实现语义约束代码生成的时候要充分利用 LLM 语言理解与生成能力^[25]，LLM 可通过对自然语言描述深度理解提取关键语义信息，再将这些关键语义信息转化为代码生成约束条件，同时还得开发对应算法和工具把约束条件嵌入代码生成过程，以此确保生成代码能准确反映自然语言描述语义内容，此外语义约束代码生成方法也需考虑代码可读性和可维护性，生成代码不仅要符合语法和语义规范，还得具备良好结构和注释以方便后续修改和扩展，借助基于 LLM 的语义约束代码生成方法能实现从自然语言描述到形式化代码高效转换^[26]，可为自动驾驶测试场景生成提供强大技术支持。

1.3.3 场景物理合理性的多模态验证机制

场景物理合理性验证在自动驾驶测试场景生成里是重要环节^[27]，它会直接影响测试场景的真实性和可靠性，为确保生成场景具备物理合理性，需要建立一种多模态验证机制来通过多种方式综合验证场景，多模态验证机制核心在于结合多种验证手段，从不同角度对场景的物理合理性展开评估，这些验证手段包括基于物理规则的验证、基于仿真数据的验证以及基于专家知识的验证等，通过多种验证手段结合能够全面评估场景物理合理性，确保生成场景符合实际交通环境物理规律^[22]，基于物理规则的验证是多模态验证机制的重要组成部分，通过定义和应用牛顿运动定律、能量守恒定律等物理规则，可对场景中物体运动和交互进行验证，比如验证车辆加速度是否符合物理规律、车辆与行人之间碰撞是否符合能量守恒等。

1.4 本文研究结构

为了达成基于自然语言输入自动生成高保真三维交通场景并做量化评估这一目标，本文构建起一个把自然语言处理、交通场景建模^[28]、三维仿真以及量化评估整合在一起的综合研究体系，本文对当前国内外在自然语言驱动的场景生成、智能仿真系统集成和自动驾驶场景评估这些方面的研究进展进行分析，从而明确研究问题以及技术挑战。

第一章绪论这部分内容阐述了研究背景以及问题提出逻辑起点，首先点明当前智能驾驶技术面临关键挑战是对低频高风险“长尾场景”识别与处理能力不足，接着分析长

尾场景典型特征包括交通参与者行为不可预测性、道路布局复杂性以及自然条件变化对系统的干扰等情况，本章还进一步指出现有测试方法难以全面覆盖这些多样化场景从而导致系统评估存在盲区，所以为实现更高保真的测试需求提出通过生成逼真三维测试场景来填补这一空白，本章节为后续系统设计与实现奠定问题基础和研究动因并明确构建智能驾驶场景生成系统的必要性与紧迫性。

第二章相关工作：这一章节围绕自然语言生成智能驾驶场景相关技术进行论述，系统介绍预训练语言模型在场景生成里的核心机制，尤其是基于 Sentence - T5 模型对自然语言场景描述做语义向量编码，再通过向量匹配去检索结构化模板，实现从语言理解到结构生成的语义桥接过程，同时深入探讨自然语言到结构化代码转换的建模逻辑，强调语义建模、语法约束建模和结构对齐的协同作用，在交通场景语义建模方面，本文提出把自然语言转化成结构化语义表示，提取参与实体、行为属性、空间关系和时序逻辑，以此生成具备真实执行逻辑的交通场景脚本，Scenic 语言作为场景描述核心工具，凭借简洁语法和概率建模能力，为多样化、高保真的场景生成提供支持，最后章节简要介绍本系统与 CARLA 智能驾驶仿真平台的集成方式，借助 Scenic 语言生成的脚本自动构建可执行三维交通场景，实现自然语言到可交互场景的完整闭环。

第三章系统需求分析与总体架构设计：主要是系统地对基于自然语言的三维智能驾驶场景生成系统做分析和设计，首先明确系统需要满足的核心功能需求，涵盖高保真自动驾驶测试场景构建需求、自然语言输入解析、语义理解与结构映射、可执行脚本生成以及仿真平台验证等关键能力，旨在以自动化方式提高智能驾驶测试效率与多样性，接着详细说明系统总体架构，采用模块化分层设计将系统划分为自然语言输入模块、结构生成模块、CARLA 仿真模块和结果评估分析模块这四大部分，其中自然语言输入模块借助图形界面与语义模型实现对用户场景描述的高质量编码与检索，结构生成模块把检索结果和预设模板相融合输出符合 Scenic 语法的结构化脚本，仿真模块基于 CARLA 平台加载脚本并自动调度执行以完成交通参与体的行为仿真与场景复现，结果评估模块对仿真过程进行图像、视频和行为数据的采集与分析并依据多维指标输出仿真效果与驾驶行为质量评估，如此便构建出一个完整的自然语言驱动智能驾驶场景生成与验证系统架构，为后续模块实现提供明确蓝图和技术支撑。

第四章系统设计与实现部分介绍系统开发环境及核心功能实现情况，硬件方面采用具备高性能的本地计算机包含 Intel i9、RTX 3060 以及 32GB 内存，软件环境是基于 Windows 11 和 Python 3.8 且集成自然语言处理与自动驾驶仿真相关依赖库，系统实现划分为三大核心模块，自然语言理解模块借助句向量检索和大模型来生成 Scenic 脚本，场景合成与仿真模块负责将脚本转化成 CARLA 里的动态三维场景，评估与展示模块对仿真效果进行可视化呈现并在语义保真度、多样性、驾驶性能等方面开展定量分析为系统验证和优化提供相应依据。

第五章系统案例测试分析：这章通过还原十组典型自然语言输入场景来展示系统在语义解析、结构化生成以及三维仿真中的完整流程和实际效果，每组案例都从文本输入开始，系统会自动完成语义理解、Scenic 脚本构建以及 Carla 场景生成，直观呈现不同

语义指令下生成结果的可视化表现，在此基础上结合语义保真度、场景多样性和系统响应效率这三大维度开展系统性能评估，实验结果显示系统在保真性和语义一致性方面最终得分能保持在 0.94 - 0.95 之间，多样性与准确性指标都达到较高水准，和规则方法以及 GPT - 4.0 方法相比展现出更好的综合生成质量和语义还原能力。

第六章总结与展望：这篇文章以自然语言处理技术为基础，设计并且实现出一种自动驾驶仿真场景生成方法，还结合 CARLA 平台达成功动态场景的语义驱动生成和感知系统的集成，系统利用深度学习模型对自然语言描述做语义解析，能够自动构建出符合语义要求的复杂交通场景，同时借助集成的目标检测与行为预测模块提高自动驾驶系统的智能决策能力，实验结果证实了该方法具备有效性和实用性，为提升自动驾驶场景多样性和感知智能化给予了有力支持。

第 2 章 相关工作

2.1 预训练语言模型在场景生成系统中的应用机制

预训练语言模型也就是 Pretrained Language Models (PLMs)，它是基于大规模语料来学习语言分布规律的通用建模方法，其核心要点是通过自监督学习掌握词序列的统计规律，并且在这个基础上实现对下游任务的泛化能力，在生成结构化代码的任务当中，本系统采用 Sentence-T5-large 作为预训练语言模型，通过 Transformer 架构实现对自然语言场景描述的语义向量编码，并基于向量相似度检索最匹配的场景模板，为后续结构化代码生成提供基础语义输入。此过程涉及对语言结构、上下文关系以及目标语言语法规则的综合建模能力。

2.1.1 基于 Sentence-T5-large 的语义编码机制在本系统中的应用

在正式开始之前我先简单介绍下 sentence-transformer 模型，它核心采用的是序列到序列 (Seq2Seq) 架构，此架构是由编码器和解码器这两部分所组成，编码器主要负责把输入句子编码成固定长度上下文向量，解码器则是利用该上下文向量去生成目标句子，这种架构最开始是用于机器翻译相关任务，不过很快就被扩展应用到其他文本生成任务当中。

为了让模型的性能得到有效提高，注意力机制被引入到 Seq2Seq 模型当中，注意力机制允许解码器在生成每个输出词时，动态地去关注输入句子里最重要的部分，而并非仅仅依赖一个固定不变的上下文向量，这种机制显著提升了模型对长句子和复杂结构的处理能力，使模型能够更精准地捕捉输入与输出之间的语义关系。

本系统在自然语言生成智能驾驶场景时采用 Sentence-T5-large 模型作为语义编码器，其作用是从自然语言输入中提取语义向量并进行高效检索，Sentence-T5 是在 Google T5 (Text-to-Text Transfer Transformer) 模型基础上微调得到的句子级别语义模型，具备对中文自然语言场景描述进行高质量语义建模的能力。

T5 模型从本质上来说是基于 Transformer 架构构建的，它的核心重点在于自注意力机制也就是 Self - Attention，此机制用于捕捉输入序列里任意两个位置之间的依赖关系，在编码过程当中，输入的自然语言描述首先会通过嵌入层也就是 Embedding 转化成向量形式，接着会在多个 Transformer 编码器层中开展上下文建模工作，在每一层里面会通过如下公式来计算注意力权重：

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V \quad (2.1)$$

其中 Q 、 K 、 V 分别代表查询 (Query)、键 (Key) 和值 (Value) 矩阵， d_k 是缩放因子。该机制能够有效建立输入序列中任意位置之间的关系，从而捕捉复杂语义、空间

关系与事件逻辑等特征。

在本系统里 Sentence - T5 - large 模型用于把用户输入中文自然语言场景描述编码成固定长度语义向量，该向量接着和预构建场景语料库中的向量做余弦相似度计算来实现语义级别检索匹配，匹配结果会当作模板驱动模块输入进而生成初步场景结构，后续再经由结构生成模块构造出符合 Scenic 语法规范的完整脚本，通过这一流程语言模型不仅完成对自然语言理解还建立起从自由描述到结构模板的桥接机制，它是本系统中自然语言到场景构建链路里的关键组件。

2.1.2 自然语言生成结构化代码的建模逻辑

把自然语言变成结构化代码的这个过程，本质上是把输入的非形式化且语义丰富自然语言映射到有严格语法和执行逻辑的目标代码语言里，这个映射包含着三个核心理论组件：

(1) 语义建模 (Semantic Modeling)：语言模型要先理解自然语言背后的意图，也就是用户描述里蕴含的操作对象、属性约束、空间关系和时间先后等语义信息，这部分依靠语言模型对上下文的长距离建模能力以及对词汇的语义表示能力。

(2) 语法约束建模 (Syntactic Constraint Modeling)：目标语言像 Scenic 这类有着明确的语法规则以及结构限制，语言模型在预训练过程里学习了多种语言模式，还能内隐地掌握特定语言的文法结构来生成合法的结构化代码。

(3) 结构对齐与表示变换 (Structure Alignment)：自然语言可以让人自由地进行表达，而结构化代码属于高度约束的形式语言。语言模型借助自回归生成的过程，把自然语言里的结构信息逐步转化成代码片段，达成从开放式表述到约束式语言的对齐映射。

需要注意的是语言模型并非通过硬编码规则做语义转换，而是凭借大规模预训练获得的统计语言知识，在解码阶段依靠概率最大化来实现语言到代码的结构生成，这种方法具备通用性强和适应性高的优势，适合用于多领域的自然语言编程任务。另外在结构生成的过程当中，可以引入显式语义约束机制或者模板结构，像代码前缀提示这类，以此进一步提升生成代码的可控性与可解释性。

2.2 自然语言到交通场景的语义建模

把自然语言描述转变为可执行的交通仿真场景，关键之处在于对语义信息做结构化建模，交通场景一般有着复杂的空间布局、多主体互动且时序性强等特征，所以要从自然语言里精准提取出参与实体、行为属性、空间关系以及时间逻辑，并且构建能被场景引擎识别和执行的语义表示结构，本节围绕场景语义结构分析与关键元素抽取、时序关系与交通意图建模这两个方面展开论述。

(1) 场景语义结构分析与关键元素抽取

交通场景里的自然语言通常会包含多个核心组成部分，比如交通参与者（像车辆、行人、自行车等）、空间约束（例如道路类型、位置关系）、行为动作（像是“行驶”“等

待”“穿越”这类)以及环境条件(比如天气、时间、信号灯状态),语义结构分析的目标是从原始语句当中建立出如下形式的结构化语义表示。

$$\text{Scene} = \{\text{Agents}, \text{Actions}, \text{Locations}, \text{Relations}, \text{Conditions}\} \quad (2.2)$$

其中: Agents 表示参与主体及其类型(如一辆蓝色轿车、自行车、行人等); Actions 表示行为描述,如“驶入交叉口”、“等待信号”、“加速通过”等; Locations 表示空间位置或背景信息,如“在红绿灯前”、“十字路口中央”、“左侧车道”; Relations 表示主体之间的空间和逻辑关系,如“位于... 左侧”、“跟随... 行驶”; Conditions 表示上下文条件,如“在夜间”、“红灯状态”、“雨天”等。

抽取过程是依靠语言模型对上下文依存关系的建模能力来开展的,同时配合命名实体识别(NER)、依存句法分析(Dependency Parsing)等语言学工具,能够形成初步的语义结构,在本系统当中,语义结构会进一步映射为 Scenic 或 Carla 里支持的场景构造要素,以此实现语义向结构化语言的桥接。

2.2.1 时序关系与交通意图建模

交通场景不只是简单的静态配置情况,还涉及高度动态化的行为序列以及意图表达内容,自然语言里经常包含清晰或者隐含的时间逻辑信息,像“接着”“在……之后”“同时”这类时间相关信息,对于还原真实交通过程起到了关键作用影响,时序关系建模的目标是将事件按照逻辑顺序组织成有向图或者序列,每个事件节点包含主体、动作以及发生的具体时间,其具体呈现形式如下:

$$\text{Timeline} = [e_1 \rightarrow e_2 \rightarrow \dots \rightarrow e_n], \quad e_i = (\text{Agent}, \text{Action}, \text{Time}) \quad (2.3)$$

借助语言模型对事件触发词以及像“当……时”“随后”这类连接词的建模能力,能够自动推理出事件之间的因果和先后关系并形成场景执行逻辑路径,另外交通意图建模所关注的是参与主体行为背后的目标导向性,比如“车辆试图避让行人”这种表述不仅描述了一个行为,还隐含着“行人优先”的交通规则与主车的决策意图,这类意图信息对于像 Carla 的行为树或触发机制这样的场景控制逻辑有着重要价值,可通过对谓词、助动词以及上下文环境进行深度理解来实现建模。

2.3 Scenic 语言的形式化语义与表达能力

Scenic 语言是专门面向智能驾驶场景的描述性语言,它主要目标是借助简洁且强大的语法结构,把复杂交通场景转化成可执行的仿真代码,为让语言具备高效表达能力和高度可扩展性,Scenic 语言将形式化的语法设计和概率建模结合起来,通过对交通场景进行准确描述,能够生成包含多主体且行为多样化的复杂仿真环境,本节会从 Scenic 的语法设计与概率建模、Scenic 在智能驾驶场景中的优势分析这两个方面展开详细讨论。

2.3.1 Scenic 的语法规设计与概率建模

Scenic 语言语法规设计是有效表达交通场景的基础，Scenic 核心在于用简单面向对象描述方式，支持对交通参与者及空间位置等灵活建模，其语法规结构支持层次化描述能精准构建场景元素，Scenic 语法规设计遵循以下几个基本原则：

面向对象建模：Scenic 通过对“对象”进行定义，像车辆、行人这类，同时也对“对象属性”加以明确，比如位置、速度、尺寸这些，以此来构建一个高层次的交通场景。

行为定义：系统支持对交通参与者的动作（像“加速”“刹车”这类动作）以及事件（例如“碰撞”这样的事件）进行定义。

空间约束与条件：Scenic 能够对空间约束进行定义，像“在道路上”这种，还能定义基于条件的约束，例如“当交通信号为红色时”。

Scenic 有个关键特性就是支持概率建模，在场景生成时可引入随机性和不确定性，通过引入概率分布，Scenic 能够模拟复杂的交通情境，生成多样化且高度不确定的场景，像车辆的初始位置、速度以及行为等，都可以用概率分布进行建模，以此更好贴近真实世界交通流的动态性。

$$P(\text{Scene}) = \int_{\Omega} P(\text{Elements}|\text{Conditions}) d\Omega \quad (2.4)$$

在此公式中， $P(\text{Scene})$ 表示生成场景的概率，Elements 代表场景中各种元素（如车辆、行人等），而 Ω 表示所有可能的场景配置空间。通过这种概率建模，Scenic 可以在多种不确定性下生成具有高度真实性和多样性的场景。

2.4 智能驾驶仿真平台 CARLA 与本系统的集成机制

本系统构建的智能驾驶场景生成平台基于 CARLA (Car Learning to Act) 作为底层仿真执行环境。CARLA 是一款开源的高保真自动驾驶模拟器，具备复杂交通环境建模、传感器模拟、物理驱动与多主体交互等功能，广泛用于自动驾驶系统的测试验证。系统中生成的结构化场景代码（基于 Scenic 表达）需要与 CARLA 无缝集成，才能完成从自然语言描述到仿真结果输出的完整闭环。

本节将重点介绍本系统如何通过 API 调用、仿真调度和代码结构设计，将生成的场景脚本自动加载并执行于 CARLA 平台。

2.4.1 本系统中的 CARLA 场景接入方式

本系统通过 Python 脚本与 CARLA 的客户端接口交互，实现自动化场景加载与仿真调度。具体接入方式包括以下三个方面：

(1) **Scenic 脚本解析与实体注入：**由自然语言生成的 Scenic 脚本将描述交通参与者的位置、动作与环境约束。本系统基于 Scenic 官方提供的解析器 ScenicSimulator，将其转化为 CARLA 所需的地图配置、初始状态及行为逻辑，并通过内部封装的 ScenicRunner 类完成实体创建与行为注入。

(2) 仿真控制与环境配置：系统调用 CARLA 的 Python API，在指定端口连接仿真服务器并设置同步模式、地图名称、天气参数、时间步长等基础环境，确保生成场景能够稳定加载和运行。仿真控制逻辑封装在‘ScenicRunnerDynamic.py’模块中，支持多场景批量运行与日志记录。

(3) 数据采集与评估接口：CARLA 提供的传感器数据流（如 RGB 摄像头、激光雷达）被系统用于场景运行效果展示与性能评估。系统支持自动截图、轨迹输出和行为评分，为后续的性能对比与优化提供基础数据。

与通用 CARLA 使用方式不同，本系统实现了从自然语言到仿真执行的端到端流程，只需输入中文场景描述，即可自动完成 Scenic 脚本生成、CARLA 场景加载与仿真结果输出，极大简化了传统场景构建与接入的开发成本。

2.4.2 Scenic 与 CARLA 的联合仿真执行流程

本系统达成了 Scenic 和 CARLA 平台的高度集成，构建出完整自然语言到仿真执行自动化流程，用户先借助图形化界面或者命令行输入中文自然语言描述，比如“自行车在路口等待红灯”这类交通场景需求，系统接收输入之后，会调用基于 Transformer 架构的 Sentence - T5 模型对输入语句做语义向量编码，还会在预构建的语料库中检索与其语义最接近的场景模板结构，以此快速确定候选场景结构以及参与元素。

语义匹配完成之后系统会基于匹配模板构造一份符合 Scenic 语法规范的场景脚本，此脚本会明确指定交通参与者类型数量初始位置行为动作及时序逻辑形成完整结构化场景表达，接着生成的 Scenic 脚本会被传入 Scenic 官方提供的 ScenicSimulator 并在本系统封装的 ScenicRunnerDynamic 模块中完成解析与实体化处理，最终将静态场景脚本转化为 CARLA 中可执行的仿真场景实例。

完成脚本加载以后系统会借助 CARLA 提供的 Python API 启动仿真环境，接着加载地图、设置天气以及创建交通主体，然后执行场景里所定义的行为，整个仿真过程会按照同步控制模式来运行，并且自动保存仿真运行期间的截图或者视频帧，同时采集轨迹、传感器数据等信息用于后续评估，通过上述模块化的处理流程，本系统达成了从自然语言到语义编码再到结构生成最后到仿真执行的端到端闭环集成，有效提升了智能驾驶场景生成的效率、语义一致性和自动化水平。

第3章 系统需求分析与总体架构设计

这章会详细介绍系统功能需求分析和总体架构设计相关内容，一开始要深入分析系统需满足的功能需求，其中涵盖自然语言输入、语义理解等关键方面，这些功能需求为系统设计提供明确方向和目标，接下来会阐述系统的总体架构设计，包含架构目标、设计原则及模块间交互机制等，借助这些内容读者可清晰了解系统整体设计，为后续章节各模块具体设计与实现奠定基础。

3.1 系统功能需求分析

3.1.1 高保真的驾驶测试需求

在自动驾驶技术快速发展的背景下，自动驾驶系统的测试与验证变得特别重要，传统测试方法通常依靠手动构建场景，这种方式不仅耗费时间精力，而且难以覆盖复杂交通场景，为提高测试效率和覆盖度，本研究提出基于自然语言描述的三维智能驾驶场景生成系统，该系统能自动把用户输入的自然语言场景描述，转化为可执行的仿真场景并在 CARLA 仿真平台验证。

本系统的一个核心优势是可利用生成的智能驾驶场景，为自动驾驶技术提供高效且低成本测试平台，传统人工构建测试场景的方法耗时又费力，还难以覆盖复杂交通场景，导致测试成本高昂且效率极其低下，相比而言本系统借助自然语言输入和自动化场景生成技术，能够快速生成多样化的驾驶场景，明显提高了测试效率，用户仅需通过简单自然语言描述来输入，系统就能自动生成对应的三维智能驾驶场景，并在 CARLA 仿真平台上进行验证，这种自动化流程极大减少了人工干预，有效降低了测试成本，除此之外系统支持多种语义表达和场景组合，能够生成丰富的测试用例，满足不同阶段自动驾驶技术测试需求，通过这种方式本系统既能加速智能驾驶技术的开发和验证过程，又能为研究人员和工程师提供灵活可扩展的测试工具，推动智能驾驶技术持续发展。

3.1.2 自然语言驱动的场景生成优势与实现

自然语言作为人类最直观的表达方式能极大降低自动驾驶测试场景构建门槛，系统运用先进的自然语言处理技术可对用户文本描述进行深入语义理解，能自动解析其中场景元素、交通参与者、环境条件及其动态行为，基于解析结果系统调用预设场景模板与动态生成模块自动构建符合描述的高保真三维驾驶场景，这种基于自然语言的自动生成方式提高了场景设计的灵活性和多样性还支持复杂场景快速搭建，像不同天气、道路结构、交通密度等多样化条件都能满足，同时系统与 CARLA 平台无缝集成实现场景即时仿真和验证方便测试人员及时调整和优化测试用例，该方法有效缩短测试准备周期提升测试质量有助于覆盖更多复杂和极端驾驶情境，为自动驾驶系统的安全性和鲁棒性提供坚实保障。

3.2 系统总体架构设计

系统整体采用模块化分层设计，包括展示层、中间层、数据层，其中本文核心重点在中间层，其主要分为三个核心模块，每个模块分有几个子模块分别负责不同阶段的处理任务，构成完整的自然语言驱动智能驾驶场景生成流程：

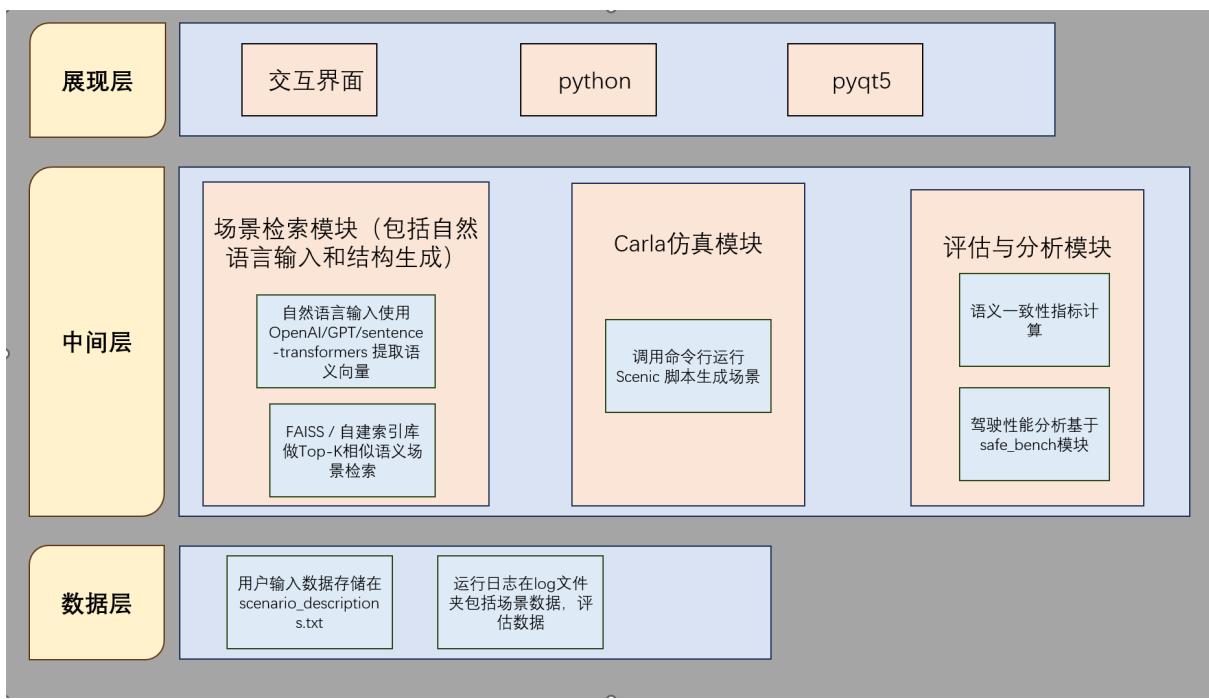


图 3-1 系统架构

3.2.1 自然语言输入模块

结合需求分析的实际情况，本系统实现了支持用户通过图形化界面或者命令行形式输入文字交通场景描述的功能，提供友好人机交互体验以保证用户能够方便输入各类场景描述，为达成这一目标，系统会运用 PyQt5 框架来构建图形化用户界面也就是 GUI，以此确保界面具有简洁高效以及跨平台兼容性的特点，PyQt5 是功能强大的 Python GUI 框架，能提供丰富控件与灵活布局选项，让开发者可以设计出直观且易于使用的界面。

具体来说系统会遵循极简主义设计原则去除不必要的复杂元素保证界面清晰直观，用户能够通过简单点击和输入操作完成场景描述输入无需复杂配置或学习，界面会提供清晰提示信息和操作指引来帮助用户快速上手使用，同时系统会优化输入流程减少用户操作步骤比如自动填充常见信息提供快捷输入选项提高输入效率，此外系统还支持用户保存输入的场景描述方便后续使用时快速调用和修改，为进一步提升用户体验系统可考虑引入智能提示功能依据用户输入内容实时推荐可能场景描述选项提高输入准确性和效率，总之系统自然语言场景输入功能以简洁高效为核心确保用户快速准确完成场景描述输入并享受流畅交互体验。

本系统在自然语言生成智能驾驶场景时采用 Sentence-T5-large 模型作为语义编码器，目的是从自然语言输入里提取语义向量并实现高效检索，Sentence-T5 是在 Google

T5 (Text-to-Text Transfer Transformer) 模型基础上微调出来的句子级别语义模型，拥有对中文自然语言场景描述进行高质量语义建模的能力，T5 模型本质上是基于 Transformer 架构构建的，其核心部分是自注意力机制（Self-Attention），作用是捕捉输入序列中任意两个位置之间的依赖关系。

3.2.2 结构生成模块

结构生成模块作为本系统连接语义理解和仿真执行的核心环节，其主要任务是把语义检索得到的场景结构模板和自然语言输入内容进行融合，进而自动生成符合 Scenic 语法规规范的结构化场景脚本也就是.scenic 文件，该模块不但要对场景中的交通参与者及其属性进行表达，还需要描述它们的行为逻辑、空间约束以及触发机制，以此保证场景具备完整性、可执行性和可控性。

在实际进行实现的时候，系统维护着一组参数化 Scenic 模板脚本，模板里面预先设置了车辆、行人、道路、红绿灯等典型交通元素的结构框架与行为函数，要是用户输入“自车前方有行人突然横穿马路并停在车前”这样的内容，系统会匹配到“横穿 - 阻挡”场景模板，还会结合输入语义给其中的参数插槽自动赋值，像横穿速度、横穿位置、停止距离等。如下：

```
""""
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.
"""

```

```

Town = globalParameters.town
EgoSpawnPt = globalParameters.spawnPt
yaw = globalParameters.yaw
egoTrajectory = PolylineRegion(globalParameters.
    waypoints)
param map = localPath(f'../maps/{Town}.xodr')
param carla_map = Town
model scenic.simulators.carla.model
EGO_MODEL = "vehicle.carlamotors.carlacola"

behavior AdvBehavior():
    do CrossingBehavior(ego, globalParameters.
        OPT_ADV_SPEED, globalParameters.OPT_ADV_DISTANCE)
        until (distance from self to egoTrajectory) <
            globalParameters.OPT_STOP_DISTANCE

```

```

while True:
    take SetWalkingSpeedAction(0)

    param OPT_GEO_X_DISTANCE = Range(2, 8)
    param OPT_GEO_Y_DISTANCE = Range(15, 50)
    param OPT_ADV_SPEED = Range(0, 5)
    param OPT_ADV_DISTANCE = Range(0, 15)
    param OPT_STOP_DISTANCE = Range(0, 1)

    ego = Car at EgoSpawnPt,
        with heading yaw,
        with regionContainedIn None,
        with blueprint EGO_MODEL

    IntSpawnPt = OrientedPoint following roadDirection
        from EgoSpawnPt for globalParameters.

        OPT_GEO_Y_DISTANCE
    AdvAgent = Pedestrian right of IntSpawnPt by
        globalParameters.OPT_GEO_X_DISTANCE,
        with heading IntSpawnPt.heading + 90 deg,
        with regionContainedIn None,
        with behavior AdvBehavior()

    require (distance from AdvAgent to intersection) > 10

```

所示的 Scenic 示例脚本中，系统使用 globalParameters 模块中的可调参数如 OPT_ADV_SPEED、OPT_GEO_X_DISTANCE 等控制行人的行为过程与生成位置，定义的行为函数 AdvBehavior() 则包含了先横穿、再停下的动态行为建模。这些参数既可来自模板设定，也可从自然语言推理中注入，最终组合成具有时间逻辑与交互动态的场景剧本。

结构生成模块能够支持在 Scenic 脚本里添加像位于路口、靠右侧车道这类空间约束以及如距离自车低于一定值停止这种行为触发条件等语义规则，同时系统也支持为多个角色赋予不同行为序列以形成多主体互动的复杂场景。

最终生成的 Scenic 脚本将自动保存至 scenario/scenario_data/scenic_data/ 目录下，供后续仿真调度模块调用执行，实现从自然语言到可执行场景的结构化表达闭环。

3.2.3 carla 仿真模块

仿真调度模块作为本系统关键组件可实现自动驾驶场景可视化与执行验证，其主要负责把生成的 Scenic 脚本加载到 CARLA 仿真环境里，同时开展仿真初始化、执行控制、传感器采集以及过程管理等工作。该模块在本系统中由封装的 ScenicRunner-Dynamic 类实现，调用 Scenic 官方提供的 ScenicSimulator 对生成脚本进行解析，并将其转化为可在 CARLA 中执行的场景实体。

在具体流程方面模块会先读取前一阶段生成的.scenic 脚本，接着通过 Scenic 编译器解析脚本里定义的交通参与者、位置、行为及约束条件，随后系统基于 Carla Python API 建立起仿真连接，配置运行地图、天气参数、物理步长（fixed_delta_seconds）与同步模式等环境变量。系统默认会采用 Carla 的同步模式来实施控制，以此保证场景里各参与体的状态更新和传感器采集过程保持一致，进而提升整个仿真过程的稳定性。

在仿真执行过程里系统会自动创建交通参与体像车辆行人等并对其行为轨迹进行初始化，要是 Scenic 脚本当中定义了行为函数例如行人横穿并在自车前停止等情况，系统就会凭借行为调度逻辑驱动对应实体去执行脚本里预定义的动作，在仿真过程期间模块还会负责记录关键执行信息包含仿真时间对象轨迹碰撞信息交通信号状态等内容，与此同时系统具备截图与视频保存功能能够自动截取仿真中间帧或者生成仿真动画用来进行结果展示与评估。

这个模块的运行过程给系统的“代码，仿真”环节提供了自动化调度能力，并且结合 CARLA 高保真的渲染与物理驱动能力，为自然语言驱动的场景生成系统提供了可视化反馈与执行验证平台，成为构成系统闭环当中的关键一环。

3.2.4 结果评估分析模块

结果展示模块是系统里用于输出仿真执行结果的重要部分，它能可视化运行过程还可辅助后续评估分析，此模块负责接收来自仿真调度模块的运行数据与视觉信息，并且会以图像视频和行为指标等多种形式呈现生成场景实际效果。

在具体实现的时候系统会在仿真执行进程中调用 CARLA 所提供的图像传感器接口，自动把关键帧截取下来并保存成静态图片的形式，也能够按照用户的设置把整个仿真过程录制成为视频文件，用来供展示和回放方面使用。与此同时系统支持记录自车以及关键参与体的轨迹坐标相关信息，形成时间序列这种形式的运动轨迹数据，可用于后续的路径可视化或者行车行为评估工作。

run_eval.py 脚本在评估执行方面起着关键作用，是实现场景质量验证和仿真效果分析的重要部分。这个模块以命令行的形式把代理配置文件和场景配置文件作为输入，可自动加载对应的强化学习模型权重，还能调用 Scenic 与 CARLA 环境完成整个仿真执行流程。在仿真的过程中，系统会详细记录自车行为和环境交互的数据，涵盖轨迹、速度、加速度以及转向变化等方面的信息，同时会监控碰撞、闯红灯、偏离路线等关键事件。

评估模块集成了一套多维度的指标体系，能够自动计算包含碰撞率、交通违规频率、路线完成度、行为稳定性、舒适性指标和安全性评分的综合性能评估结果。所有的

评估数据会统一输出成为日志文件，并且支持以图像或者视频形式记录结果，方便开发者进行回放和可视化分析。系统还内置了自动路径管理机制，确保每轮评估过程的日志和图像结果能够有序存储，有利于进行实验对比和复现。

借助该评估模块，系统能够从自然语言场景生成开始，完整实现仿真、记录、评估的一体化闭环流程，显著提升对生成场景合理性和策略效果的量化验证能力。

第4章 系统设计与实现

4.1 开发环境

4.1.1 硬件环境

这个实验是靠性能比较强的本地计算机来完成的，它主要的硬件配置情况如下所示，处理器选用的是 Intel Core i9 这款产品，它具备多核心以及高主频的特性，能够支持复杂计算任务和高并发程序运行，显卡采用的是 NVIDIA GeForce RTX 3060 这一型号，具备较强的图形渲染与并行计算方面能力，有利于运行自动驾驶模拟平台比如 CARLA 0.9.13 和进行深度学习模型推理，内存大小达到了 32GB 这样的容量，大容量内存保证了在多线程和大数据处理时系统流畅性，能有效避免出现内存瓶颈方面的相关问题，这样的硬件配置为自然语言场景生成、自动驾驶仿真以及模型评估等任务提供稳定且高效运行环境。

4.1.2 软件环境

系统开发和运行所基于的操作系统是 Windows 11，其为微软最新发布出来的操作系统，能提供良好的用户体验以及强大的系统性能，并且和各类开发工具与依赖库具备广泛兼容性，它支持多种编程语言以及开发框架，可满足项目开发过程里的各种需求。

项目主要是将 Python 3.8 作为开发语言来运用，Python 属于一种被广泛使用的高级编程语言，它凭借简洁的语法以及强大的库支持而闻名于世，在自然语言处理、机器学习和自动驾驶仿真等诸多领域均有广泛应用，Python 3.8 能够提供稳定的语言特性和相关优化，可满足项目开发过程中的各类实际需求。

软件环境需要的依赖库方面已经安装好，具体如下表所示：

库名称	版本
openai	最新版本
sentence_transformers	最新版本
torch	1.13.1+cu117
torchvision	0.14.1+cu117
torchaudio	0.13.1
gym	0.23.1
numpy	1.21.6
pygame	2.3.0
tqdm	4.65.0
pyyaml	6.0
matplotlib	3.5.3
opencv-python	4.7.0.72
pandas	1.5.3
seaborn	0.12.2
shapely	1.8.5
ephem	4.1.4
joblib	1.2.0
cpprb	10.7.0
pycocotools	2.0.6
moviepy	1.0.3
scikit-image	0.19.3
transformers	最新版本
setGPU	最新版本

表 4-1 项目依赖库及其版本

4.2 系统核心功能实现

4.2.1 自然语言理解功能

在自然语言输入这方面首先要了解到的就是测试数据，其是为了综合体现自动驾驶场景多样复杂特点，涵盖像直行障碍、转弯障碍等多种典型场景类别，例如直行障碍、转弯障碍、变道、超车、闯红灯、无保护左转、右转以及交叉路口协商等，这些场景类别覆盖自动驾驶车辆实际道路环境可能遇到的关键交互情况，具备较高代表性和实用价值。数据集中场景描述以自然语言形式进行呈现，并且包含一定比例模糊性描述，以此模拟真实世界里驾驶员或其他交通参与者行为不可预测性，场景描述模糊性设计有助于测试生成系统应对不精确指令时的处理能力，确保系统能够适应并生成符合要求的复杂交通场景。

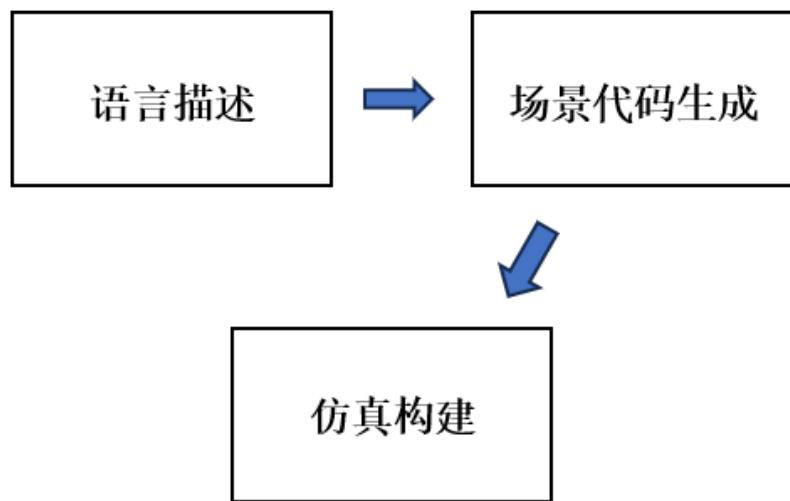


图 4-1 从自然语言描述到仿真场景构建的流程图

本模块负责接收自然语言输入并生成对应的 Scenic 场景描述。具体流程如下：

输入：自然语言形式的场景描述，例如 “The ego vehicle is driving on a straight road; the adversarial pedestrian suddenly appears from behind a parked car on the right front and suddenly stop.”。
检索增强：使用 sentence-transformers 中的 sentence-t5-large 模型对输入进行向量化表示，并在本地检索数据库（如 retrieve/scenario_descriptions.txt）中查找相似描述作为参考样本。
大语言模型解析：采用预训练的大语言模型（如 GPT-4o），结合检索到的参考样本，生成符合输入语义的 Scenic 脚本。
场景拼接机制：根据场景复杂度，支持对多个子元素（如车辆、行人、环境条件）进行组织与拼接。

4.2.2 场景合成与仿真模块

本模块承担着把自然语言生成的 Scenic 脚本转化成三维可视化仿真场景的任务，并且要在 CARLA 仿真平台上达成功态运行，整个流程涵盖脚本解析、场景构建、仿真控制、交互支持以及数据输出等多个环节，具体情况如下：

Scenic 解析：本系统会先借助 Scenic 内置的语法与语义分析器来对输入脚本开展解析工作，Scenic 作为一种领域专用语言可让用户用简洁且结构化的方式去描述场景元素像车辆、行人、障碍物等的初始状态和约束条件，解析过程能够生成涵盖对象属性如位置、速度、朝向以及交互规则如跟车距离、避让行为还有环境条件如天气、时间的完整场景配置

仿真构建：完成脚本解析之后系统自动把 Scenic 生成中间配置映射到 CARLA 仿真平台里，具体涵盖载入指定地图、部署交通参与者、设定摄像头和 LiDAR 等传感器参数、配置天气光照路况等环境因素等内容，系统能够依据配置实现对城市道路高速公路十字路口等多种类型场景精准建模接口调用关系：Scenic 借助和 CARLA 集成的 Python

API 来调用底层仿真接口，系统会自动完成 Actor 的创建以及状态初始化和行为脚本绑定等任务，这极大程度简化了从脚本到仿真的转换流程，研究人员不用手动编程就能通过文本控制仿真流程，显著提升了效率和复用性。

动态交互支持：系统能够支持高度动态化的场景仿真工作，还可以模拟交通参与者之间的交互行为，像车辆变道、避障、红绿灯响应以及行人横穿马路等情况，Scenic 语言提供了条件触发与时间控制相关机制，允许用户描述复杂的行为序列内容，以此实现具有时序性、可演化的场景变化状况，从而增强了仿真的真实感与复杂程度。

多样化仿真配置模块能灵活进行仿真参数配置，涵盖地图选择像 Town01、Town05 等类型，包含天气设置有晴天、雨天、雾天等情况，还有交通流量控制以及车辆控制模式如自动驾驶、手动控制等内容，可满足不同测试需求，且该模块支持不同版本 CARLA 和 Scenic 组合使用，具备较好兼容性与可拓展性。

输入数据：本实验采用约 20 条自然语言描述作为输入样本，用于生成自动驾驶场景。这些描述覆盖多种交通情境和突发事件，例如：

- (1) 自车在红绿灯前等待信号
- (2) 行人突然横穿街道
- (3) The ego vehicle is driving on a straight road; the adversarial pedestrian appears from a driveway on the left and suddenly stop and walk diagonally.
- (4) The ego vehicle is driving on a straight road when a pedestrian suddenly crosses from the right front and suddenly stops as the ego vehicle approaches.

下面我用一个具体的例子展示我是如何实现 carla 场景生成的，**场景描述：**自车在直路上行驶时，一名行人突然从右前方穿出

(自然语言输入) The ego vehicle is driving on a straight road when a pedestrian suddenly crosses from the right front and suddenly stops as the ego vehicle approaches.

Listing 4.1: 场景关键代码示例

```

1   behavior AdvBehavior():
2       do CrossingBehavior(ego, globalParameters.
3           OPT_ADV_SPEED, globalParameters.OPT_ADV_DISTANCE
4           )
5       until (distance from self to egoTrajectory) <
6           globalParameters.OPT_STOP_DISTANCE
7       while True:
8           take SetWalkingSpeedAction(0)
9
10      ego = Car at EgoSpawnPt,
11          with heading yaw,

```

```

9      with regionContainedIn None,
10     with blueprint EGO_MODEL
11
12     IntSpawnPt = OrientedPoint following roadDirection
13     from EgoSpawnPt
14     for globalParameters.OPT_GEO_Y_DISTANCE
15     AdvAgent = Pedestrian right of IntSpawnPt by
16       globalParameters.OPT_GEO_X_DISTANCE,
17     with heading IntSpawnPt.heading + 90 deg,
18     with regionContainedIn None,
19     with behavior AdvBehavior()
20
21   require (distance from AdvAgent to intersection) >
22     10

```



图 4-2 场景截图

调用生成的 `scenic` 代码，实现仿真模拟脚本，以及场景保存。

4.2.3 场景评估与展示模块

场景评估与展示模块的目的是对生成的三维仿真场景做可视化呈现和定量分析，此模块方便研究人员直观了解场景生成具体效果，也为后续性能比较、模型调优以及系统验证提供评估依据，该模块包含下面几个关键功能：

可视化展示：在仿真进行的过程当中系统能够自动截取关键帧的截图，或者录制完整的仿真过程视频并保存成图像或视频文件，截图一般会选择交通事件发生点像刹车、避障、碰撞等情况，或者特定的时序节点，以此确保所展示信息具备代表性与丰富性，

视频部分可以借助 CARLA 原生录制功能或者集成第三方渲染引擎来实现，并且支持多角度、多摄像机的可视观察，从而进一步增强场景调试与验证方面的可操作性。

评价指标主要涵盖安全性、驾驶性能、场景语义一致性和系统鲁棒性等多个方面，安全性指标包含碰撞次数、闯红灯次数、交通规则违章等内容，用来衡量系统的安全保障能力，驾驶性能指标诸如轨迹偏差、车道保持率和速度控制平稳性，反映车辆行驶的准确性和舒适度，场景语义一致性指标评估生成场景与自然语言描述的匹配度及多样性，系统鲁棒性指标关注系统在异常情况下的恢复能力及对复杂环境的适应性，综合这些指标能够全面量化自动驾驶系统在不同测试场景中的表现并指导系统优化与改进。

量化评估：本系统是在自动化生成以及仿真的基础之上，进一步引入多维度的量化评估指标，以此对场景生成的有效性、合理性和多样性进行定量测量，具体涵盖的内容有：

(1) **语义保真度 (Semantic Fidelity)**: 这个指标的作用是衡量输入的自然语言描述和最终生成的仿真场景之间语义一致性，可采用人工标注评分和自动化匹配算法相结合的方式来开展，比如通过设定关键语义要素像地点、交通行为、天气条件等的匹配程度，对场景是否准确还原用户意图进行评分，自动方法能够结合自然语言处理与图结构匹配技术实现初步评估进而提升效率。

(2) **多样性指标 (Scene Diversity)**: 这项指标能反映系统生成场景在多次输入不同自然语言之后的差异性以及覆盖范围，主要涵盖空间多样性像不同地图位置与道路类型等方面、元素多样性比如参与车辆行人非机动车种类情况、行为多样性包含速度控制路线选择交通交互等内容，统计方法有使用 Shannon 熵、Jaccard 相似度或者聚类分布指标等，以此全面反映生成系统的泛化能力和表现范围。

(3) **驾驶性能指标 (Driving Performance)**: 为了评估生成场景对自动驾驶系统测试价值，本模块支持在生成场景里运行预设自动驾驶控制系统，像基于 CARLA 的自动驾驶 Agent 并记录其表现，关键评估维度包含碰撞率也就是单位场景中发生碰撞的比例、任务成功率即是否成功完成任务或驶出场景、路径偏离率指与理想路径的偏差程度等，这些数据能够用于评估生成场景的挑战性与安全测试覆盖度，是场景质量评估的重要依据。

评估结果展示与导出：系统将展示代码运行出的评估结果用列出的形式来做可视化展示，所有截图、评估指标还有分析数据都可以导出成标准格式文件，例如 CSV、JSON、PNG 等，可用于论文撰写、模型调试或者进一步的数据挖掘。

表 4-2 系统评估结果

指标名称	数值
碰撞率 (collision_rate)	0.0
红灯违规频率 (avg_red_light_freq)	0.0
停车标志违规频率 (avg_stop_sign_freq)	0.0
出路面长度 (out_of_road_length)	0.0
路线跟踪稳定性 (route_following_stability)	0.806
路线完成度 (route_completion)	0.93
平均时间消耗 (avg_time_spent)	4.5
平均加速度 (avg_acceleration)	0.32
平均偏航角速度 (avg_yaw_velocity)	0.0
车道入侵频率 (avg_lane_invasion_freq)	0.0
安全操作得分 (safety_os)	1.0
任务完成得分 (task_os)	0.802
舒适度得分 (comfort_os)	1.0
最终综合得分 (final_score)	0.941

在这个示例评估里系统展现出较高安全性，碰撞率和交通违规频率两个数据均为零，这表明系统在基础安全保障方面表现相当不错，路线跟踪稳定性达到大约 0.81 这个数值，显示系统具备较好的路径跟踪能力，路线完成度或许意味着按预定路线行驶情况良好，安全操作和舒适度的得分都拿到了满分，说明车辆运行状态平稳且符合安全标准，最终综合得分是 0.94 这个成绩，整体表现算得上优秀但仍有提升空间，尤其是在任务完成度这一方面。

4.3 系统编码与实现

4.3.1 系统整体架构与主程序设计

本系统基于 Python 语言开发，结合 Carla 0.9.13 与 SafeBench 框架，整体架构模块化且层次清晰。系统以两个主控脚本 `run_train.py` 和 `run_train_dynamic.py` 作为入口，分别负责仿真评估与动态场景训练。参数统一通过 `argparse` 管理，涵盖实验配置、设备选取、模型路径、输出目录等，保障了实验的灵活性和复现性。

在整体的运行流程当中，系统会先依据传入的配置文件去加载代理（Agent）和场景参数，之后按照策略类型自动选择对应的运行器（Runner）并启动仿真过程，同时在设计里充分考虑了配置文件路径的动态拼接以及输出目录的自动创建，以此确保文件管理的规范与安全。

```
# 伪代码：主程序执行流程
parse_args()
```

```

for each agent_cfg in agent_cfg_list:
    for each scenario_cfg in scenario_cfg_list:
        set_device_and_seed()
        agent_config = load_config(agent_cfg)
        scenario_config = load_config(scenario_cfg)
        setup_output_dirs()
        if scenario_config.policy_type == 'scenic':
            runner = ScenicRunner(agent_config, scenario_config)
        else:
            runner = CarlaRunner(agent_config, scenario_config)
        runner.run()

```

4.3.2 主控脚本功能与运行流程

`run_train.py` 支持三种主要模式：代理训练 (`train_agent`)、场景训练 (`train_scenario`) 和评估 (`eval`)，并依据配置自动调用 `CarlaRunner` 或基于 `Scenic` 场景语言的 `ScenicRunner`。该设计方便用户针对不同实验需求进行灵活切换。

而 `run_train_dynamic.py` 则专注于依据自然语言描述开展动态场景训练，默认加载动态场景的配置文件，实现自然语言到 `Scenic` 代码的实时转换，自动创建和训练相关的模型保存路径，以此确保训练过程的中断恢复与结果管理。两者均利用多线程控制计算资源，支持 GPU 设备切换，并可选渲染和视频保存，为仿真结果的展示和后期分析提供便利。

4.3.3 动态场景仿真控制与异常处理

动态场景仿真系统的关键模块，基于 `ScenicRunner` 的扩展实现。系统会接收自然语言所做的描述，然后经过预处理与解析的步骤，将其转换成 `Scenic` 语言脚本内容，接着由运行器对脚本进行加载并执行，最终自动生成符合语义要求的三维交通场景。

在仿真流程里系统会对每个场景实例做初始化，载入车辆、传感器以及任务配置等内容，并且严格把控仿真时长和步长，为保障系统具备健壮性引入了异常捕获机制，可捕获并打印详细异常堆栈信息，还能保证后续实验连续执行。

```

# 伪代码：动态场景仿真异常处理
try:
    runner.run(test_epoch)
except Exception as e:
    runner.close()
    log_error(traceback.format_exc())
    continue_with_next_experiment()

```

本系统借助合理设计出来的主程序结构、配置管理以及动态场景运行机制，达成了从自然语言描述到三维智能驾驶仿真的高效转换，为智能驾驶仿真研究和场景生成提供强有力的技术支撑。

第 5 章 系统案例测试分析

这一节主要是要展示系统在自然语言驱动交通场景生成任务里的实际运行效果，接下来我会对案例的分析测试进行拓展，从纵向和横向这两个不同方面去分析生成的效果，通过选取多个有代表性的输入示例，这些示例分别涵盖不同类型的交通语义指令，像静态车道场景、红绿灯交叉口、雨天行车以及复杂车辆交汇等情况，以此全面体现系统在语义理解、场景构建、三维仿真和评估反馈等方面的功能表现。

每一个示例均从用户界面输入开始，系统首先对自然语言进行语义编码与理解，然后通过检索模块筛选相关的语义片段或模板，接着由生成模块构造结构化的 Scenic 脚本，最终由 Carla 仿真环境还原为三维可视化场景。系统自动捕捉仿真结果图像，并结合评估模块生成语义一致性、多样性、结构完整性等量化指标，综合展示该场景的构建质量。

5.1 场景案例测试

为了验证系统生成场景是否有多样性和高保真性，本章节会展示若干个典型的测试场景，这些场景包含红绿灯等待和夜间行驶等真实交通情境，每个场景都是通过自然语言输入生成而来，并且会在 CARLA 仿真平台当中运行获得图像结果，通过这些实例能直观观察系统对不同语义描述解析能力，以及最终生成场景所呈现出的真实感与复杂度。

5.1.1 场景一：自车在道路上行驶时，一名行人突然从左侧前方穿出

用户自然语言输入：The ego vehicle is driving on a road when a pedestrian suddenly crosses from the left front and suddenly stops as the ego vehicle approaches.



图 5-1 行人从左侧前方穿出 - 视角一



图 5-2 行人从左侧前方穿出 - 视角二

从横向对比来看，每一次场景生成，都会有一定的变动与误差，在这个场景中，自车与行人的相对位置在生成的时候会有差异。分析原因可能有，当车速与行人的相对速

度过大时会造成生成出来的效果偏差。

5.1.2 场景二：两辆自行车在直路上前后行驶

用户自然语言输入：Two ego vehicles are driving in a straight line, one following the other.



图 5-3 两辆自行车直线行驶 - 视角一



图 5-4 两辆自行车直线行驶 - 视角二

在本次对比中可以看出，视角的选取不同，也会影响效果的差异，所以调用相机的模块在场景生成中也十分重要。

5.1.3 场景三：一名行人横穿马路而自行车在直路上行驶

用户自然语言输入：A ego vehicle is driving on a straight road while a pedestrian is crossing the street.



图 5-5 行人横穿马路 - 视角一



图 5-6 行人横穿马路 - 视角二

5.1.4 场景四：夜晚行人在人行道上走

用户自然语言输入：At night, a pedestrian is walking on the sidewalk.

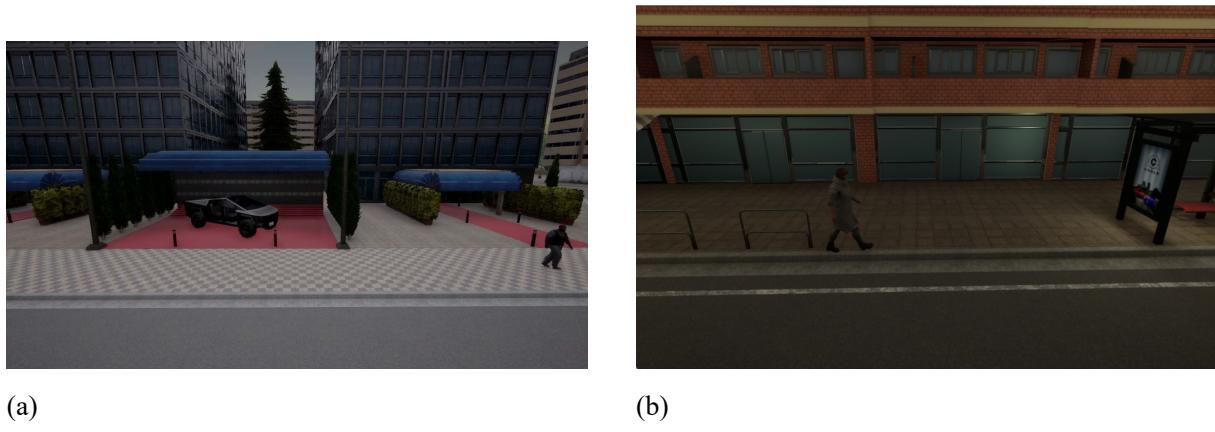


图 5-7 夜晚行人在人行道上走

该场景展示了横向对比中生成的行人运动状态也会有所不同，证明对目标主体的状态描述可以更加详尽。

5.1.5 场景五：自车通过十字路口

用户自然语言输入：The ego vehicle passes through the intersection.

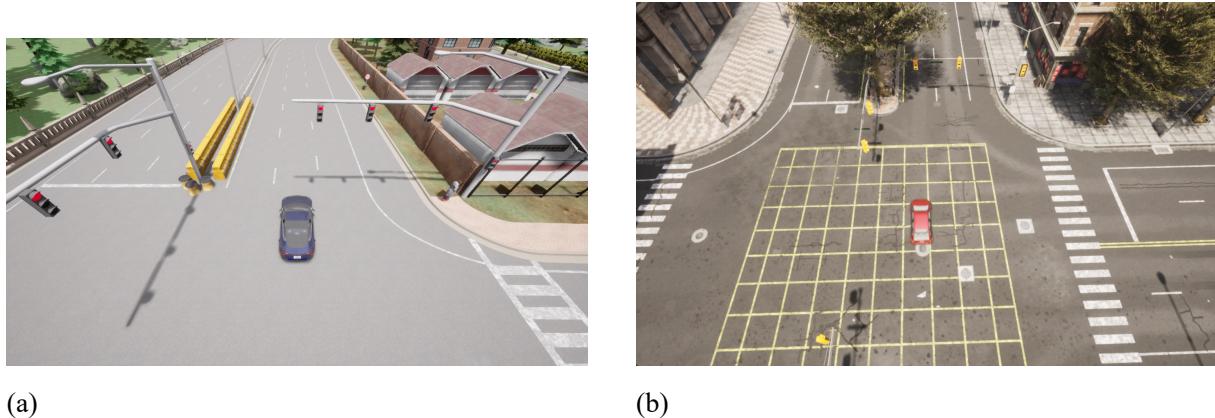


图 5-8 自车通过十字路口

场景五展示了 town1 和 town5 中运行代码的效果。

5.2 评估测试分析

为全面、客观地衡量本系统在从自然语言生成高保真三维交通场景过程中的表现，本文从语义保真度、场景多样性与系统效率三个核心维度构建了一套评估体系，具体指标设计如下：

(1) 语义保真度 (Semantic Fidelity)：它主要是评估生成场景能不能准确还原输入自然语言里的核心语义信息，以此确保系统生成结果在语义层面有一致性和完整性。评估内容包含参与主体的一致性，像描述里提到的车辆类型比如红色轿车、卡车，还有行

人、障碍物是否出现在仿真场景当中，空间位置关系，例如“在路口等待”“位于右侧车道”“靠近斑马线”等描述是否体现在实体布局里面，行为逻辑一致性，比如“等待绿灯”“直行通过”“与前车保持车距”等行为是否在仿真中能体现出来。评估方式采用检索式语义匹配评分也就是自动化加上人工评审打分也就是主观验证的双重手段，自动化方面是用自然语言处理模型计算输入描述和生成场景之间的语义相似度，主观方面是由3名评估者依据统一评分标准对每条样本进行1到5分打分，然后取平均作为最终得分。

(2) 场景多样性 (Scene Diversity): 多样性评估用于衡量系统在面对不同自然语言输入时所生成场景之间的差异性，防止生成内容趋于模板化或重复模式。主要从以下几个角度度量：

1. 结构多样性 (Structure Diversity Score, SDS): 通过提取场景中的车辆、行人、建筑等静态元素的布局特征，计算不同场景之间的结构差异程度；

2. 行为多样性 (Behavioral Diversity Score, BDS): 通过仿真轨迹分析车辆在不同行为状态（加速、刹车、等待、避让等）上的变化情况，评估其动态多样性；

3. 地图覆盖率方面要统计不同输入生成场景在 CARLA 地图里的分布情况，以此评估是否存在特定区域集中度过高的问题，采用像结构差异率、行为状态熵等定量指标来开展自动化评估工作，并且结合统计图表做可视化展示。

(3) 系统效率与响应能力 (Efficiency): 系统效率是衡量该平台在真实应用场景下可行性的重要指标，重点考察系统从接收自然语言指令到输出完整三维场景所需的总时间开销，包括：检索耗时：Sentence-T5 向量化及相似描述查找所用时间；生成耗时：大语言模型生成 Scenic 脚本所耗时间；仿真加载与渲染时间：Scenic 编译后至 CARLA 仿真运行启动所经历的延迟；总响应时间：综合上述所有子流程，从输入自然语言到仿真画面出现的端到端时间。

通过日志记录还有脚本打点的方式来采集各个阶段的耗时，并且用平均响应时长 (ms)、方差这类指标对性能表现进行量化，系统在多轮连续输入情况下的响应稳定性也被纳入评估范围。

在评估生成场景的质量时本研究选两种有代表性方法作基线对比，基于规则的方法即 Rule - based 这种方法通过定义明确逻辑和规则生成场景，它具有较好可解释性适合处理结构化和预定义任务，不过面对复杂和模糊的自然语言描述时灵活性和适应性较差，基于 GPT - 4.0 的方法借助大型语言模型生成能力来生成丰富多样的场景，该方法在场景生成多样性方面具备优势，不过在生成场景的物理合规性和逻辑一致性上存在不足，尤其是在处理复杂交通情境的时候。选择这两种基线的主要目的是从不同角度评估本研究方法在生成场景质量、效率以及针对复杂和模糊指令的鲁棒性方面所具备的优势。

平均得分为 4.38，说明生成系统在语义还原方面表现优异，能够较好地捕捉自然语言中的关键词及其空间/行为语义。

5.2.1 准确率分析

在输入条件近似的情况下，系统生成了具有多样参与者布局与动作的场景。采用结构特征编码后计算场景间欧几里得距离，多样性指标如碰撞率、闯红灯频率、停车标志频率、超出道路长度、路线跟随稳定性、路线完成率、平均花费时间、平均加速度、平均偏航速度以及车道入侵频率等，下面两张图通过横向对比，同一场景生成的结果分析会有所不同，但最后得出 final Score 保持在 0.94-0.95 之间，说明系统具备良好的准确率5-1。此外，通过对生成的截图进行视觉对比，发现系统在车辆类型、行驶方向、光照与天气等维度的变化也具备一定随机性与可控性。以下为两个评估测试的结果截图：

Evaluation Metric	Value
Collision Rate (碰撞率)	0.0
Avg. Red Light Frequency (平均闯红灯频率)	0.0
Avg. Stop Sign Frequency (平均停车标志频率)	0.0
Out-of-Road Length (超出道路长度)	0.0
Route Following Stability (路线跟随稳定性)	0.8954
Route Completion (路线完成率)	0.9932
Avg. Time Spent (平均花费时间)	6.0342
Avg. Acceleration (平均加速度)	0.687
Avg. Yaw Velocity (平均偏航速度)	0.0
Avg. Lane Invasion Frequency (平均车道入侵频率)	0.0
Safety OS (安全操作系统)	1.0
Task OS (任务操作系统)	0.6863
Comfort OS (舒适操作系统)	1.0000
Final Score (最终得分)	0.9519

表 5-1 Evaluation Results1

Evaluation Metric	Value
Collision Rate (碰撞率)	0.0
Avg. Red Light Frequency (平均闯红灯频率)	0.0
Avg. Stop Sign Frequency (平均停车标志频率)	0.0
Out-of-Road Length (超出道路长度)	0.0
Route Following Stability (路线跟随稳定性)	0.8542
Route Completion (路线完成率)	0.9865
Avg. Time Spent (平均花费时间)	5.325
Avg. Acceleration (平均加速度)	0.7869
Avg. Yaw Velocity (平均偏航速度)	0.0
Avg. Lane Invasion Frequency (平均车道入侵频率)	0.0
Safety OS (安全操作系统)	1.0
Task OS (任务操作系统)	0.6934
Comfort OS (舒适操作系统)	1.0000
Final Score (最终得分)	0.9409

表 5-2 Evaluation Results2

5.2.2 效率分析

系统整体生成流程的平均时间如下：

为了评估基于自然语言生成高保真智能驾驶仿真场景系统的整体性能，我们把完整流程划分成五个主要的阶段，并且针对每一阶段的平均运行时间、关键输出结果以及其影响因素进行了详细统计与分析，具体内容如表 5-3 所示。

阶段	平均时间 (秒)	关键输出	影响因素 / 可调参数
语义检索与匹配	1.6	相似场景向量 + 匹配索引	检索库规模、向量维度、 匹配阈值、Top-K 设置
Scenic 脚本生成	2.3	Scenic 场景脚本 (.scenic)	场景模板复杂度、语言理解精度、地形元素数量
Scenic 到配置转换	0.7	.config 配置文件	转换规则数量、是否包含动态目标
Carla 场景加载与构建	3.2	Carla 世界状态	地图大小、天气设置、对象密度
截图渲染与保存	1.1	RGB 图像 (.png)	分辨率、渲染设备性能、 截图角度
总耗时	8.9	—	与系统性能、并发线程数等有关

表 5-3 自然语言生成场景的各阶段性能统计

首先语义检索与匹配阶段的主要任务是把用户输入的自然语言描述转化成向量表示，然后和预先构建好的场景语义向量库进行相似度检索，这个阶段平均耗费时间为 1.6 秒，输出的内容是一组相似场景的索引以及向量，其主要受到检索库规模、向量维度、匹配阈值还有 Top - K 值设置等因素的影响。

其次在 Scenic 脚本生成阶段会把检索到的匹配结果和语言模板相结合，进而自动构造出符合语义的.scenic 脚本，此阶段平均耗时为 2.3 秒，该阶段的耗时会受到场景描述复杂程度、地形构造模板数量、语言解析精度等因素的影响，特别是在涉及多个车辆与动态目标的情况下，脚本生成时间会有所上升。

接下来进入 Scenic 到配置转换这个阶段，此阶段会把.scenic 文件解析成 Carla 兼容的.config 配置文件，以此为后续仿真环境加载做好准备，这一过程平均需要花费 0.7 秒时间，处理逻辑相对来说比较轻量，转换规则的具体数量以及是否包含动态对象像自动驾驶目标车辆、行人等会略微对耗时产生影响。

第四阶段在整个流程里是最耗时间的部分，具体为 Carla 场景加载与构建，此阶段涵盖场景地图加载、静态和动态对象放置、交通信号配置等操作，平均所花费的时间是 3.2 秒，该阶段的性能很大程度上依赖于地图的大小、环境的复杂程度（像是否包含天气、光照等因素）以及硬件图形性能。

最后，截图渲染与保存阶段会调用 Carla 客户端 API 截取仿真视角下 RGB 图像并保存，此阶段平均耗时为 1.1 秒，影响因素包含图像分辨率、摄像机角度以及渲染硬件性能等。从整体情况来看整个从自然语言输入到仿真图像输出流程平均耗时大概 8.9 秒，在单线程运行模式下已经具备比较不错的响应性能，要是进一步采用并发优化或者异步渲染技术，该系统在未来能够扩展成支持实时交互与多轮问答生成的智能仿真平台。经验证，在一次性处理 20 条自然语言输入的情形下，系统可在不显著增加响应时间的前提下完成全部场景的构建与仿真，平均每个场景的处理时间波动控制在 ± 0.7 秒范围内。这表明：

- (1) 系统核心模块间的异步处理与缓存机制有效；
- (2) 大语言模型调用延迟已通过缓存和批处理策略优化；
- (3) Scenic 与 CARLA 接口连接在高频调用场景下仍保持稳定。

总体来说系统拥有良好响应效率和可扩展性，不但支持单条交互式输入方式，而且能够满足批量生成相关需求，适合大规模自动驾驶测试等实际应用任务，比如仿真训练或者仿真场景库构建等，若要进一步提升系统的运行效率，可通过这些途径来进行优化：引入本地轻量化大模型以减少云端调用延迟；对检索模块进行并行化改造，提升向量查找速度；使用 CARLA 的异步仿真接口或地图预加载技术降低启动时间。

5.3 本章小节

这章通过多个典型自然语言驱动交通场景案例，系统且全面地展示本系统在语义理解、场景生成、三维仿真和评估反馈等方面整体能力，实验结果表明系统能够准确捕捉输入指令中的核心语义信息，生成具有高保真度的三维交通场景，并且在车辆行为、环

境光照、行人动作等细节上呈现出良好多样性与合理性，经过语义保真度和多样性两大核心指标量化评测，系统在语义一致性和场景丰富性方面都取得了优异成绩。

另外系统在响应效率方面体现出较强实时性与稳定性，平均单条场景生成时间可控制在 9 秒以内，同时还支持批量输入处理，比较适合大规模自动驾驶仿真应用，通过基于规则方法和基于大语言模型方法对比验证，进一步突出本系统在灵活性和复杂场景处理上的优势。总的来说本章测试分析充分证明本系统在自然语言到三维智能驾驶场景自动生成领域有效性和实用价值，给后续系统优化和应用推广奠定了坚实基础。

第6章 总结与展望

6.1 工作总结

目前自动驾驶技术正处于快速且持续的发展阶段，在复杂交通环境当中借助高效场景生成与感知系统去提升自动驾驶系统智能性和安全性成为研究关键要点，本文设计并实现基于自然语言处理的自动驾驶仿真场景生成方法，结合 Carla 仿真平台探讨利用自然语言输入生成动态仿真场景并结合感知系统提高自动驾驶决策能力，本研究围绕“自然语言驱动的自动驾驶场景生成与感知”这一问题展开，采用基于预训练大模型的检索与生成方法利用深度学习技术提升场景生成灵活性和多样性，通过对输入自然语言描述开展语义分析让系统能够自动生成符合语义要求的场景并在 Carla 平台上进行仿真验证，除此之外本文还设计集成感知系统通过深度学习与图像处理技术实现交通目标检测、跟踪与行为预测为后续决策与控制提供支持。

实验结果显示基于自然语言生成的仿真场景可精准反映输入描述语义，还能借助感知系统实时跟踪并预测交通目标行为意图，该系统能在复杂交通环境中稳定运行，这验证了自然语言描述和自动驾驶场景生成具有可行性与有效性。本文的贡献是提出一种创新性的自动驾驶场景生成与感知方法，通过自然语言生成仿真场景并结合感知与决策支持，提升了自动驾驶系统的适应性与智能化水平，研究中采用的 Carla 仿真平台为系统验证与优化提供了丰富测试数据，未来研究成果有望为智能驾驶系统多样化场景生成和感知能力提升提供更多技术支持。

6.2 研究不足

虽然本文在自然语言生成场景和感知系统研究方面取得了一定进展，不过在某些方面依旧存在着不足与局限性，当前的场景生成方法虽然能够生成基本的交通场景，但是处理更为复杂的动态环境如极端天气和特殊道路条件时就显得力不从心，现有的自然语言理解模型对于复杂描述的理解能力和场景生成的精确度有待提升，面对不常见的交通情境生成的场景缺乏丰富性和真实感，尽管本文的感知系统在多数情况下能够准确跟踪和预测目标行为，但是在高密度交通或目标交错的情况下会出现目标跟踪失误从而影响行为预测准确性，在多目标竞态和复杂背景下现有的目标跟踪算法仍可能受到影响进而导致系统长期跟踪出现问题。

虽说本研究把深度学习和物理建模结合起来做行为意图分析，可现有的方法依靠物理模型简单规则，或许难以捕捉更复杂多变的驾驶行为，特别是多个目标之间存在复杂交互的时候，系统可能没办法准确预测其行为，最后，系统在处理高分辨率图像和复杂场景时，其实时性和计算性能还存在一定瓶颈，虽说现有系统能满足基本实时性要求，但在高负载条件下，响应时间和处理速度可能会受影响，这在实际应用当中也许会带来挑战。

6.3 后续优化方向

在未来的研究当中系统的优化会集中在下面这几个方面首先提升场景生成的多样性和复杂度会是未来研究的重点内容当前系统生成的交通场景主要是以基本场景作为主体面对复杂交通情况的时候依然存在着一定的不足未来的研究将会采用更先进自然语言处理和场景生成模型结合强化学习等相关方法提高系统在极端天气事故等复杂情况中的应对能力，进而生成更加多样化且真实的场景其次目标跟踪与行为意图预测的精度会得到进一步的提升虽然现有的跟踪算法在大多数情况之下能够保持目标稳定跟踪但是在高密度或者复杂背景的状况下仍然存在一定精度问题。

未来的工作会进一步探索更加鲁棒的目标跟踪算法结合深度学习和多传感器融合技术提高系统的目标识别和跟踪能力此外意图预测会结合更多像交通规则和社会行为这样的情境因素以此来提升对复杂驾驶行为的预测精度。

再者系统的实时性和计算性能会持续不断做优化，毕竟随着场景复杂度逐渐增加系统实时性和处理能力成关键问题，未来研究将重点聚焦于高效图像处理和计算方法，要借助硬件加速以及算法优化来提高系统计算效率，以此确保在高负载和复杂场景下系统依旧可以保持流畅运行，最后未来研究要结合真实道路测试与仿真验证，把研究成果转化为实际可应用的具体内容，通过在真实环境中开展测试进一步评估系统性能，确保系统在多样化且复杂交通环境中具备稳定性，通过以上这些方面的优化未来系统能够更好应对动态复杂交通场景，从而提高自动驾驶系统智能感知能力，为实现更高安全性和智能化水平的自动驾驶技术奠定基础。

参考文献

- [1] ABEYSIRIGOONAWARDENA Y, SHKURTI F, DUDEK G. Generating Adversarial Driving Scenarios in High Fidelity Simulators[C]//2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019: 8271-8277.
- [2] BAGSCHIK G, MENZEL T, MAURER M. Ontology Based Scene Creation for the Development of Automated Vehicles[C]//2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2018: 1813-1820.
- [3] BIGGIO B, CORONA I, MAIORCA D, et al. Evasion Attacks against Machine Learning at Test Time[C]//Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2013: 387-402.
- [4] BROWN T, MANN B, RYDER N, et al. Language Models are Few-Shot Learners[J]. Advances in Neural Information Processing Systems, 2020, 33: 1877-1901.
- [5] CAI P, LEE Y, LUO Y, et al. SUMMIT: A Simulator for Urban Driving in Massive Mixed Traffic[C]//2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020: 4023-4029.
- [6] CUI J, LI Z, YAN Y, et al. Chatlaw: Open-source legal large language model with integrated external knowledge bases[J]. arXiv preprint arXiv:2306.16092, 2023.
- [7] CHOWDHERY A, NARANG S, DEVLIN J, et al. Palm: Scaling language modeling with pathways[J]. arXiv preprint arXiv:2204.02311, 2022.
- [8] FENG S, YAN X, SUN H, et al. Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment[J]. Nature Communications, 2021, 12(1): 748.
- [9] KONG Z, GUO J, LI A, et al. PhysGAN: Generating Physical-World-Resilient Adversarial Examples for Autonomous Driving[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, 2020: 14254-14263.
- [10] KLISCHAT M, LIU E I, HOLTKE F, et al. Scenario Factory: Creating Safety-Critical Traffic Scenarios for Automated Vehicles[C]//2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). IEEE, 2020: 1-7.
- [11] XU Z, ZHANG Y, XIE E, et al. DriveGPT4: Interpretable end-to-end autonomous driving via large language model[J]. arXiv preprint arXiv:2310.01412, 2023.
- [12] CONTRIBUTORS S R. Carla Scenario Runner[EB/OL]. 2019. https://github.com/carla-simulator/scenario_runner.

- [13] 武彪,任洪泽,郑联庆,等.基于自然驾驶行为的智能驾驶复杂场景构建方法[J].华南理工大学学报(自然科学版),2025,53(02): 38-47.
- [14] 顾同成,徐东伟,孙成巨.无人驾驶深度强化学习决策模型性能评测方法综述[J/OL].计算机工程与应用,2025: 1-42. <http://kns.cnki.net/kcms/detail/11.2127.TP.20250414.1614.026.html>.
- [15] 伍毅平,张鸿鹏,陈家源,等.考虑场景与驾驶风格差异的出租车驾驶人驾驶行为特性分析[J/OL].武汉理工大学学报(交通科学与工程版),2025: 1-7. <http://kns.cnki.net/kcms/detail/42.1824.U.20250319.1733.016.html>.
- [16] 彭海洋,计卫星,刘法旺.基于多节点仿真的自动驾驶场景数据保护方法[J].汽车工程学报,2025: 1-12.
- [17] 方虹苏,常城,石鑫雨,等.自动驾驶场景中 YOLO 目标检测算法的应用研究[J].汽车实用技术,2025, 50(06): 65-74.
- [18] 朱宇,徐志刚,赵祥模,等.基于 TsGAN 的自动驾驶汽车高速公路变道切入测试场景自动生成算法[J].华南理工大学学报(自然科学版),2024, 52(08): 76-88.
- [19] WANG J, PUN A, TU J, et al. AdvSim: Generating safety-critical scenarios for self-driving vehicles[C]// Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021: 9909-9918.
- [20] XU C, DING W, LYU W, et al. Safebench: A benchmarking platform for safety evaluation of autonomous vehicles[C]// Advances in Neural Information Processing Systems: vol. 35. 2022: 25667-25682.
- [21] YANG Z, CHAI Y, ANGUELOV D, et al. SurfelGAN: Synthesizing realistic sensor data for autonomous driving[C]// Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 11118-11127.
- [22] YAO S, ZHAO J, YU D, et al. React: Synergizing reasoning and acting in language models[C]// The Eleventh International Conference on Learning Representations. 2022.
- [23] ZHANG L, PENG Z, LI Q, et al. CAT: Closed-loop adversarial training for safe end-to-end driving[C]// Conference on Robot Learning. 2023: 2357-2372.
- [24] ZHANG Q, HU S, SUN J, et al. On adversarial robustness of trajectory prediction for autonomous vehicles[J]. arXiv preprint arXiv:2201.05057, 2022.
- [25] ZHENG L, CHIANG W L, SHENG Y, et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena[J]. arXiv preprint arXiv:2306.05685, 2023.
- [26] ZHONG Z, REMPE D, CHEN Y, et al. Language-guided traffic simulation via scene-level diffusion[J]. arXiv preprint arXiv:2306.06344, 2023.

- [27] 赵祥模, 童星, 穆柯楠, 等. 面向自动驾驶汽车测试需求的关键场景要素提取方法[J]. 中国公路学报, 2025: 1-15.
- [28] 杜德慧, 叶振, 郑成行, 等. 面向自动驾驶系统的场景建模及边缘关键场景生成[J]. 软件学报, 2025: 1-19.

附录 A 附录：系统模块源代码

A.1 主要代码

run__train__dynamic.py: 作用：用于在动态生成的场景上训练代理。该文件使用 dynamic__scenic.yaml 进行配置，并运行训练过程，优化代理的行为。

```
import setGPU
import traceback
import os
import os.path as osp

import torch
from safebench.util.run_util import load_config
from safebench.util.torch_util import set_seed,
    set_torch_variable
from safebench.carla_runner import CarlaRunner
from safebench.scenic_runner_dynamic import
    ScenicRunner

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--exp_name', type=str, default='exp')
    parser.add_argument('--output_dir', type=str, default='log')
    parser.add_argument('--ROOT_DIR', type=str, default=osp.abspath(osp.dirname(osp.dirname(osp.realpath(__file__)))))

    parser.add_argument('--max_episode_step', type=int,
        default=300)
```

```

parser.add_argument('--auto_ego', action='store_true')
parser.add_argument('--mode', '-m', type=str, default=
    'eval', choices=[ 'train_scenario', 'train_agent', 'eval'])
parser.add_argument('--agent_cfg', nargs='*', type=str,
    , default=[ 'adv_scenic.yaml'])
parser.add_argument('--scenario_cfg', nargs='*', type=
    str, default=[ 'dynamic_scenic.yaml'])
parser.add_argument('--continue_agent_training', '-cat',
    , type=bool, default=False)
parser.add_argument('--continue_scenario_training', '-cst',
    , type=bool, default=False)

parser.add_argument('--seed', '-s', type=int, default
    =0)
parser.add_argument('--threads', type=int, default=4)
parser.add_argument('--device', type=str, default='
    cuda:0' if torch.cuda.is_available() else 'cpu')

parser.add_argument('--num_scenario', '-ns', type=int,
    default=1, help='num of scenarios we run in one
    episode')
parser.add_argument('--save_video', action='store_true'
    )
parser.add_argument('--render', type=bool, default=
    True)
parser.add_argument('--frame_skip', '-fs', type=int,
    default=1, help='skip of frame in each step')
parser.add_argument('--port', type=int, default=2002,
    help='port to communicate with carla')
parser.add_argument('--tm_port', type=int, default
    =8002, help='traffic manager port')
parser.add_argument('--fixed_delta_seconds', type=
    float, default=0.1)
args = parser.parse_args()
args_dict = vars(args)

```

```

err_list = []
for agent_cfg in args.agent_cfg:
    for scenario_cfg in args.scenario_cfg:
        # set global parameters
        set_torch_variable(args.device)
        torch.set_num_threads(args.threads)
        set_seed(args.seed)

        # load agent config
        agent_config_path = osp.join(args.ROOT_DIR, 'safebench'
            '/agent/config', agent_cfg)
        agent_config = load_config(agent_config_path)

        # load scenario config
        scenario_config_path = osp.join(args.ROOT_DIR, 'safebench/scenario/config',
            scenario_cfg)
        scenario_config = load_config(scenario_config_path)

        agent_config['load_dir'] = osp.join(agent_config['load_dir'],
            'dynamic_scenario')
        # Check if the directory exists; if not, create it
        if not osp.exists(agent_config['load_dir']):
            os.makedirs(agent_config['load_dir'])

        # main entry with a selected mode
        agent_config.update(args_dict)
        args_dict['output_dir'] = osp.join('log', 'adv_train',
            args.mode, agent_config['policy_name'], f'{agent_cfg.split('.')[0]}',
            "dynamic_scenario")
        scenario_config.update(args_dict)
        scenario_config['num_scenario'] = 1 ### 'the
            num_scenario can only be one for scenic'
        runner = ScenicRunner(agent_config, scenario_config)

        # start running
        runner.run()

```

```

for err in err_list:
    print(err[0], err[1], 'failed!')
    print(err[2])

```

`dynamic_scenic.yaml`: 作用: 该文件是代理的配置文件, 包含了代理的训练设置, 包括其行为模型、对抗性行为、场景属性等。它与其他 YAML 文件结合使用来指定代理在不同场景中的行为。

```

policy_type: 'scenic'
scenario_category: 'scenic'

route_dir: 'safebench/scenario/scenario_data/route'
scenic_dir: 'safebench/scenario/scenario_data/
    scenic_data/'
sample_num: 50
opt_step: 10
select_num: 2

method: 'scenic'
scenario_id: null
route_id: [0,1,2,3,4,5,6,7]

ego_action_dim: 2
ego_state_dim: 4
ego_action_limit: 1.0

```

`run_eval\dynamic.py`: 该文件用于运行评估流程, 调用 `evaluate_scene_quality.py` 对生成的场景进行评估。

```

import setGPU
import traceback
import os.path as osp

```

```

import torch

from safebench.util.run_util import load_config
from safebench.util.torch_util import set_seed,
    set_torch_variable
from safebench.carla_runner import CarlaRunner
from safebench.scenic_runner_dynamic import
    ScenicRunner

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--exp_name', type=str, default='
        exp')
    parser.add_argument('--output_dir', type=str, default='
        log')
    parser.add_argument('--ROOT_DIR', type=str, default=
        osp.abspath(osp.dirname(osp.dirname(osp.realpath(
            __file__)))))

    parser.add_argument('--max_episode_step', type=int,
        default=300)
    parser.add_argument('--auto_ego', action='store_true')
    parser.add_argument('--mode', '-m', type=str, default='
        eval', choices=[ 'train_agent', 'train_scenario', '
        eval'])
    parser.add_argument('--agent_cfg', nargs='*', type=str,
        default=[ 'adv_scenic.yaml'])
    parser.add_argument('--scenario_cfg', nargs='*', type=str,
        default=[ 'dynamic_scenic.yaml'])
    parser.add_argument('--continue_agent_training', '-cat',
        type=bool, default=False)
    parser.add_argument('--continue_scenario_training', '-cst',
        type=bool, default=False)

    parser.add_argument('--seed', '-s', type=int, default
        =0)

```

```

parser.add_argument('--threads', type=int, default=4)
parser.add_argument('--device', type=str, default='cuda:0' if torch.cuda.is_available() else 'cpu')

parser.add_argument('--num_scenario', '-ns', type=int,
                    default=2, help='num of scenarios we run in one
episodes')
parser.add_argument('--save_video', action='store_true')
parser.add_argument('--render', type=bool, default=True)
parser.add_argument('--frame_skip', '-fs', type=int,
                    default=1, help='skip of frame in each step')
parser.add_argument('--port', type=int, default=2002,
                    help='port to communicate with carla')
parser.add_argument('--tm_port', type=int, default
                    =8002, help='traffic manager port')
parser.add_argument('--fixed_delta_seconds', type=
                    float, default=0.1)
parser.add_argument('--test_policy', type=str, default
                    ='sac')
parser.add_argument('--test_epoch', type=int, default=
                    None)
args = parser.parse_args()

err_list = []
for agent_cfg in args.agent_cfg:
    for scenario_cfg in args.scenario_cfg:
        # set global parameters
        set_torch_variable(args.device)
        torch.set_num_threads(args.threads)
        set_seed(args.seed)

        # load agent config
        agent_config_path = osp.join(args.ROOT_DIR, 'safebench
            /agent/config', agent_cfg)
        agent_config = load_config(agent_config_path)

```

```

agent_config[ 'policy_name' ] = args.test_policy

## load the corresponding model ##
agent_config[ 'load_dir' ] = osp.join(agent_config[ 'load_dir' ], "dynamic_scenario")

# load scenario config
scenario_config_path = osp.join(args.ROOT_DIR, 'safebench/scenario/config', scenario_cfg)
scenario_config = load_config(scenario_config_path)

args.output_dir = osp.join('log', 'adv_train', args.mode, agent_config[ 'policy_name' ], f'{agent_cfg.split('.')[0]}_epoch{args.test_epoch}')
args.exp_name = "dynamic_scenario"
args_dict = vars(args)
# main entry with a selected mode
agent_config.update(args_dict)
print(agent_config[ 'load_dir' ])
scenario_config.update(args_dict)

scenario_config[ 'num_scenario' ] = 1 # the num_scenario can only be one for scenic
runner = ScenicRunner(agent_config, scenario_config)

# start running
try:
    runner.run(args.test_epoch)
except:
    runner.close()
    traceback.print_exc()
    err_list.append([agent_cfg, scenario_cfg, traceback.format_exc()])

for err in err_list:
    print(err[0], err[1], 'failed!')
    print(err[2])

```

```

run\eval.py
import setGPU
import traceback
import os.path as osp

import torch

from safebench.util.run_util import load_config
from safebench.util.torch_util import set_seed,
    set_torch_variable
from safebench.carla_runner import CarlaRunner

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--exp_name', type=str, default='exp')
    parser.add_argument('--output_dir', type=str, default='log')
    parser.add_argument('--ROOT_DIR', type=str, default=osp.abspath(osp.dirname(osp.dirname(osp.realpath(__file__)))))
    parser.add_argument('--max_episode_step', type=int, default=300)
    parser.add_argument('--auto_ego', action='store_true')
    parser.add_argument('--mode', '-m', type=str, default='eval', choices=['train_agent', 'train_scenario', 'eval'])
    parser.add_argument('--agent_cfg', nargs='*', type=str, default=['adv_scenic.yaml'])
    parser.add_argument('--scenario_cfg', nargs='*', type=str, default=['eval_scenic.yaml'])
    parser.add_argument('--continue_agent_training', '-cat', type=bool, default=False)

```

```

parser.add_argument('--continue_scenario_training', '--cst', type=bool, default=False)

parser.add_argument('--seed', '-s', type=int, default=0)
parser.add_argument('--threads', type=int, default=4)
parser.add_argument('--device', type=str, default='cuda:0' if torch.cuda.is_available() else 'cpu')

parser.add_argument('--num_scenario', '-ns', type=int, default=2, help='num of scenarios we run in one episode')
parser.add_argument('--save_video', action='store_true')
parser.add_argument('--render', type=bool, default=True)
parser.add_argument('--frame_skip', '-fs', type=int, default=1, help='skip of frame in each step')
parser.add_argument('--port', type=int, default=2002, help='port to communicate with carla')
parser.add_argument('--tm_port', type=int, default=8002, help='traffic manager port')
parser.add_argument('--fixed_delta_seconds', type=float, default=0.1)
parser.add_argument('--test_policy', type=str, default='sac')
parser.add_argument('--route_id', type=int, default=0)
parser.add_argument('--scenario_id', type=int, default=0)
parser.add_argument('--test_epoch', type=int, default=None)
args = parser.parse_args()

err_list = []
for agent_cfg in args.agent_cfg:
    for scenario_cfg in args.scenario_cfg:
        # set global parameters

```

```

set_torch_variable(args.device)
torch.set_num_threads(args.threads)
set_seed(args.seed)

# load agent config
agent_config_path = osp.join(args.ROOT_DIR, 'safebench'
    /agent/config', agent_cfg)
agent_config = load_config(agent_config_path)
agent_config['policy_name'] = args.test_policy

## load the corresponding model ##
agent_config['load_dir'] = osp.join(agent_config['
    load_dir'], f'scenario_{args.scenario_id}')

# load scenario config
scenario_config_path = osp.join(args.ROOT_DIR, '
    safebench/scenario/config', scenario_cfg)
scenario_config = load_config(scenario_config_path)
scenario_config['scenario_id'] = args.scenario_id

args.output_dir = osp.join('log', 'adv_train', args.
    mode, agent_config['policy_name'], f'{agent_cfg.
        split('.')[0]}_epoch{args.test_epoch}', f'{{
            scenario_cfg.split('.')[0]}}')

args.exp_name = 'scenario_' + str(scenario_config['
    scenario_id'])

args_dict = vars(args)
# main entry with a selected mode
agent_config.update(args_dict)
print(agent_config['load_dir'])
scenario_config.update(args_dict)
if scenario_config['policy_type'] == 'scenic':
    from safebench.scenic_runner import ScenicRunner
    scenario_config['num_scenario'] = 1 # 'the
        num_scenario can only be one for scenic'
    scenario_config['route_id'] = [args.route_id]
runner = ScenicRunner(agent_config, scenario_config)

```

```

else:
    ## id shift due to the test settings in safebench v1
    scenario_config['route_id'] = args.route_id + 4
    runner = CarlaRunner(agent_config, scenario_config)

# start running
try:
    runner.run(args.test_epoch)
except:
    runner.close()
    traceback.print_exc()
    err_list.append([agent_cfg, scenario_cfg, traceback.format_exc()])

```

```

for err in err_list:
    print(err[0], err[1], 'failed!')
    print(err[2])

```

run__train.py

```

import sys
print(sys.path)

import setGPU
import traceback
import os
import os.path as osp

import torch
from safebench.util.run_util import load_config
from safebench.util.torch_util import set_seed,
    set_torch_variable
from safebench.carla_runner import CarlaRunner

if __name__ == '__main__':

```

```

import argparse
parser = argparse.ArgumentParser()
parser.add_argument('--exp_name', type=str, default='exp')
parser.add_argument('--output_dir', type=str, default='log')
parser.add_argument('--ROOT_DIR', type=str, default=osp.abspath(osp.dirname(osp.dirname(osp.realpath(__file__))))
parser.add_argument('--max_episode_step', type=int, default=300)
parser.add_argument('--auto_ego', action='store_true')
parser.add_argument('--mode', '-m', type=str, default='eval', choices=['train_scenario', 'train_agent', 'eval'])
parser.add_argument('--agent_cfg', nargs='*', type=str, default=['adv_scenic.yaml'])
parser.add_argument('--scenario_cfg', nargs='*', type=str, default=['train_scenario_scenic.yaml'])
parser.add_argument('--continue_agent_training', '-cat', type=bool, default=False)
parser.add_argument('--continue_scenario_training', '-cst', type=bool, default=False)

parser.add_argument('--seed', '-s', type=int, default=0)
parser.add_argument('--threads', type=int, default=4)
parser.add_argument('--device', type=str, default='cuda:0' if torch.cuda.is_available() else 'cpu')

parser.add_argument('--num_scenario', '-ns', type=int, default=1, help='num of scenarios we run in one episode')
parser.add_argument('--save_video', action='store_true')
parser.add_argument('--render', type=bool, default=

```

```

    True)
parser.add_argument('--frame_skip', '-fs', type=int,
                    default=1, help='skip of frame in each step')
parser.add_argument('--port', type=int, default=2002,
                    help='port to communicate with carla')
parser.add_argument('--tm_port', type=int, default
                    =8002, help='traffic manager port')
parser.add_argument('--fixed_delta_seconds', type=
                    float, default=0.1)
parser.add_argument('--scenario_id', type=int, default
                    =None)
args = parser.parse_args()
args_dict = vars(args)

err_list = []
for agent_cfg in args.agent_cfg:
    for scenario_cfg in args.scenario_cfg:
        # set global parameters
        set_torch_variable(args.device)
        torch.set_num_threads(args.threads)
        set_seed(args.seed)

        # load agent config
        agent_config_path = osp.join(args.ROOT_DIR, 'safebench'
                                     '/agent/config', agent_cfg)
        agent_config = load_config(agent_config_path)

        # load scenario config
        scenario_config_path = osp.join(args.ROOT_DIR, 'safebench/scenario/config', scenario_cfg)
        scenario_config = load_config(scenario_config_path)

## modification
if args.scenario_id:
    scenario_config['scenario_id'] = args.scenario_id

    agent_config['load_dir'] = osp.join(agent_config[',

```

```

    load_dir'], f"scenario_{scenario_config['scenario_id']}"])

# Check if the directory exists; if not, create it
if not osp.exists(agent_config['load_dir']):
    os.makedirs(agent_config['load_dir'])

# main entry with a selected mode
agent_config.update(args_dict)
args_dict['output_dir'] = osp.join('log', 'adv_train',
                                   args.mode, agent_config['policy_name'], f'{agent_cfg.split('.')[0]}',
                                   f"scenario_{scenario_config['scenario_id']}")

scenario_config.update(args_dict)
if scenario_config['policy_type'] == 'scenic':
    from safebench.scenic_runner import ScenicRunner
    scenario_config['num_scenario'] = 1 ### 'the
                                         num_scenario can only be one for scenic'
    runner = ScenicRunner(agent_config, scenario_config)
else:
    runner = CarlaRunner(agent_config, scenario_config)

# start running
runner.run()

for err in err_list:
    print(err[0], err[1], 'failed!')
    print(err[2])

```

致谢

今天凝结在这篇毕业论文里的心血与思考，离不开一路以来给予我帮助和鼓励的每一位良师益友，在此我用最诚挚的谢意，向所有支持我陪伴我成长的人致以深深感谢。

首先要最衷心地感谢我的导师王海东老师，在论文撰写的整个过程当中，王老师始终都给予我悉心指导和极大信任，从课题选题与研究方向的确立，到每一阶段方案的推进和每一段文字的打磨，王老师都投入大量心血且不厌其烦为我解答困惑，王老师严谨治学的态度、开阔的学术视野和包容的为人风范深深影响着我，也会成为我未来求学与科研道路上的精神坐标。

感谢一起学习的同学以及朋友们，在项目开发、技术调试还有论文撰写的各个阶段，都为我提供了无私的帮助，不管是模型部署时经历的一次次失败重启，还是凌晨时分大家共享代码与想法进行的头脑风暴，这段一起研究共同进步的日子，让我既收获了珍贵友谊也得到了成长。

我还要特别感谢我的家人们，感谢你们始终如一地给予我理解、支持并且默默地守护着我，当我深夜坐在桌前专注敲击代码，遇到挫折感到焦虑的时候，你们给予我温暖和坚定的力量，是我最为坚强的后盾，你们的宽容与支持，是我能够全身心投入学术探索的前提和底气。

同时我要感谢自己在这段旅程里坚持与投入，从自然语言学习到三维场景生成探索，从理解科研理论到完成系统实现工作，我首次真切领略到科研严谨与创造力结合，这也进一步坚定我在人工智能与智能交通领域深耕志向。

最后感谢这个时代给予我施展个人才华的舞台，感谢母校提供了优质教育资源与开放氛围，让我有机会接触前沿技术并勇敢探索前行。