



本科毕业设计(论文)



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

专 业: 大数据与人工智能

班 级: 数智 2102 班

指导老师: 王海东 讲师

人工智能与先进计算学院

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 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

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	1
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	4
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 本文研究框架	5
第 2 章 理论基础	8
2.0.1 预训练语言模型与代码生成机制	8
2.0.2 自然语言到交通场景的语义建模	9
2.0.3 Scenic 语言的形式化语义与表达能力	10
2.0.4 智能驾驶仿真平台 CARLA 与集成机制	11
2.0.5 本文方法的创新点与理论意义	12
第 3 章 总体架构设计	14
3.1 系统设计目标	14
3.1.1 高保真性	14
3.1.2 自动化	14
3.1.3 可扩展性	14
3.1.4 核心功能实现	15
3.1.5 系统目标的长期愿景	15
3.2 系统总体架构	15
3.3 主要模块功能设计	16
3.3.1 自然语言理解与场景生成模块	16
3.3.2 场景合成与仿真模块	17
3.3.3 场景评估与展示模块	18

第 4 章 生成场景的质量验证	19
4.1 实验设置	19
4.2 生成效果展示(示例)	20
4.2.1 场景一: 摩托车和汽车在红绿灯前等待信号	20
4.2.2 场景二: 自我车辆在夜晚穿越道路	20
4.2.3 场景三: 自我车辆在夜晚红绿灯前等待信号	21
4.2.4 场景四: 行人横穿直行道路	22
4.2.5 更多	23
4.3 实验总结	26
第 5 章 场景生成的量化评估	28
5.1 评估指标设计	28
5.2 实验设置	29
5.3 评估结果与分析	31
5.3.1 语义保真度	31
5.3.2 准确率分析	31
5.3.3 效率分析	32
第 6 章 总结与展望	33
6.1 工作总结	33
6.2 研究不足	33
6.3 后续优化方向	34
致谢	35
附录 A 附录: 系统模块源代码	36
A.0.1 其他	61
参考文献	36

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，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，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，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。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

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，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



图 1-1 车辆样图

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，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，其次大语言模型生成的代码质量较高，它能学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，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，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，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，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，**Chen et al. (2022)** 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。

挑战与进展方面大语言模型生成场景代码有一定进展但仍面临挑战，首先处理自然语言中歧义和模糊性是个难题，复杂场景描述下生成场景可能无法完全符合预期，其次现有模型生成能力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，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，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，**Xie et al. (2021)** 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，例如，**Zhao et al. (2020)** 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，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。**Dai et al. (2020)** 我这边提出一种基于关键帧截取的可视化方法，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，**Dai et al.(2020)** 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，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。**Li et al. (2021)** 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，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，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，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，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。

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，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，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，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，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，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 本文研究框架

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、三维仿真以及量化评估整合在一起的综合研究体系，整体研究框架情况如下，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。

1. 基于大语言模型的 Scenic 场景代码生成技术

研究怎样借助预训练大语言模型像 GPT - 4o 结合检索增强机制如 Sentence - T5 来解析自然语言指令并生成符合语义的交通场景描述脚本 Scenic，重点解决生成脚本在语义准确性以及交通合理性方面的问题。

2. 交通场景代码到三维仿真的合成机制

研究怎样把自然语言生成的场景脚本高效转化成可运行的三维智能驾驶仿真场景，通过集成 Scenic 语言和 CARLA 仿真平台实现车辆行人环境等交通要素在虚拟世界构建与动态演化

3. 生成场景的展示与量化评估方法

建立一套多维度的评估体系来对生成场景进行分析，从语义保真度、多样性和驾驶性能这三个方面量化考量生成场景的质量与有效性，设计自动化的指标计算方法并且结合视觉展示手段，以此增强实验的可解释性。

在前面提到的研究基础之上本文完成系统整体实现以及实验验证，通过验证证实所提出方法具备有效性，同时总结本研究创新点与潜在改进空间。

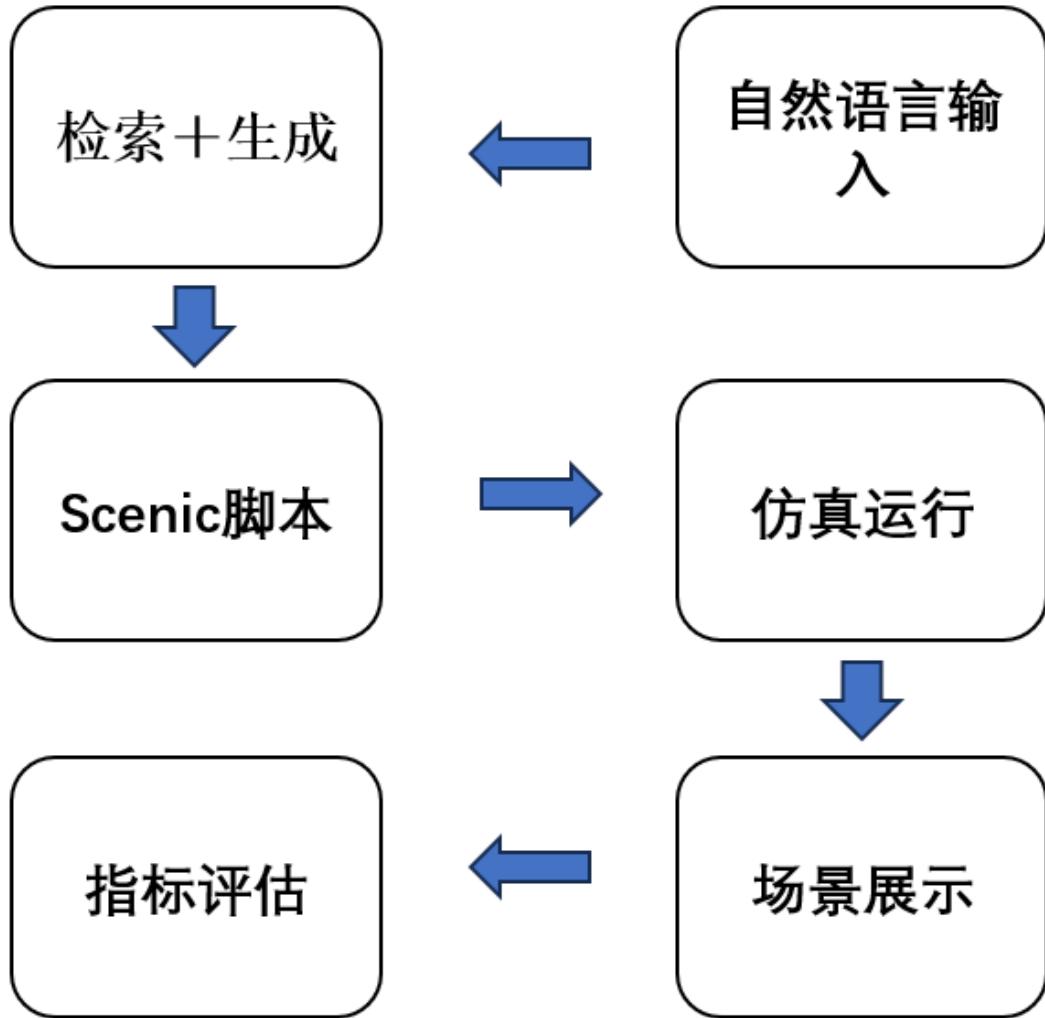


图 1-2 本文的研究框架

第 2 章 理论基础

2.0.1 预训练语言模型与代码生成机制

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，预训练语言模型会先理解自然语言的语义意图，然后将其映射为具有严格语法和语义约束的目标语言（像 Scenic）的代码表示，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。

(1) Transformer 与大语言模型的理论机制

Transformer 是现在主流语言模型所采用的基础结构，其核心是自注意力机制也就是 Self - Attention，主要用来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，输入序列首先会通过嵌入也就是 Embedding 层转化成向量表示，接着会通过多层的自注意力网络开展上下文建模，在每一层里面，模型会依据 Query - Key - Value 结构来计算注意力权重：

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

在这其中 Q 、 K 、 V 分别表示查询 (Query)、键 (Key) 和值 (Value) 矩阵， d_k 为维度归一化因子。这个机制可以在任意位置之间建立联系，从而有效捕捉长距离依赖。

在生成模型当中像 GPT 系列这类，Transformer 被当作自回归结构来使用，当前 token 的表示要依赖之前所有 token 的上下文，目标是去预测下一个 token 出现的概率，经过大规模语料的训练后模型能够学习到语言里的统计规律、语法模式以及推理结构

(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当中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。

2.0.2 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一般会包含多个核心组成部分，像交通参与者（例如车辆、行人、自行车等）、空间约束（比如道路类型、位置关系）、行为动作（像是“行驶”“等待”“穿越”）以及环境条件（例如天气、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**: 参与主体及其类型（如一辆蓝色轿车、自行车、行人等）；
- **Actions**: 行为描述，如“驶入交叉口”、“等待信号”、“加速通过”等；
- **Locations**: 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**: 主体之间的空间和逻辑关系，如“位于... 左侧”、“跟随... 行驶”；
- **Conditions**: 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) 时序关系与交通意图建模

交通场景可不只是静态配置这么简单，还涉及高度动态化的行为序列以及意图表达，自然语言里常常包含清晰或者隐含的时间逻辑，像“接着”“在……之后”“同时”这类信息，对于还原真实交通过程起着关键作用，时序关系建模的目标是把事件按照逻辑顺序组织成一个有向图或者序列，每个事件节点包含主体、动作以及发生时间，具体形式如下：

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

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，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。

2.0.3 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。

(1) Scenic 的语法设计与概率建模

Scenic 语言语法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，其语法结构支持层次化描述能精准构建场景元素，Scenic 语法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 的关键特性之一是支持概率建模，允许在场景生成时引入随机性和不确定性。通过概率分布的引入，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。例如，车辆的初始位置、速度、甚至行为都可以通过概率分布进行建模，以更好地贴近真实世界中交通流的动态性。

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

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) Scenic 在智能驾驶场景中的优势分析

Scenic 语言的设计给智能驾驶领域的场景生成带来明显优势，下面从多个维度分析 Scenic 在智能驾驶仿真里的独特优势：

- **高效的场景描述与复用：** Scenic 凭借高度抽象化的语法以及简洁的模型表达，极大程度提高了交通场景的描述效率，场景具备很强的重用性，不同场景能够通过微小的语法变更来进行扩展和组合，可满足智能驾驶测试里对多样化场景的需求。
- **动态行为与时序控制：** Scenic 可以在场景里嵌入复杂的时序关系，能准确模拟多个主体之间的互动情况，就像车辆与行人之间的避让行为，又或是车辆在不同时间点的动作序列，都能够依靠简洁的代码来实现，这让 Scenic 成为了模拟交通意图和决策过程的重要工具。
- **概率建模提升真实性：** 通过引入随机性与不确定性，Scenic 能够模拟出更贴近真实世界的交通场景，在生成多个交通参与者行为的时候，场景生成不局限于确定性条件，而是可包含行为模式的随机变化，这对测试自动驾驶系统的适应性十分重要。
- **与仿真平台的兼容性：** Scenic 语言和多种智能驾驶仿真平台像 CARLA 有良好兼容性，借助 Scenic 语言用户能生成符合平台要求的场景代码，可直接和仿真环境对接开展测试工作，避免传统场景构建里繁琐的人工编码操作。
- **可扩展性与灵活性：** Scenic 语言能够生成基于规则的固定场景，还能借助引入外部因素、环境变化以及多主体交互等元素扩展出更复杂场景，不管是常规的城市街道、环路，还是极端天气条件、突发事件等情况都可通过 Scenic 进行灵活建模。

2.0.4 智能驾驶仿真平台 CARLA 与集成机制

智能驾驶仿真平台 CARLA 也就是 Car Learning to Act，它是一款开源的高保真自动驾驶模拟器，在自动驾驶系统的开发和测试方面有着广泛应用。CARLA 具备丰富的环境建模功能，能够模拟各种天气、时间、交通和驾驶情况。正因为如此，它成了自动驾驶仿真里不可或缺的工具之一。本节会介绍 CARLA 的场景接入机制，还会阐述其与 Scenic 的联合仿真流程设计。

(1) CARLA 场景接入机制

CARLA 的场景接入机制主要是和外部代码或者仿真系统做对接，用户借助 CARLA 的 API 能够自定义并且加载复杂的交通场景，还可在这个基础上开展仿真测试，具体的

接入方式包含如下内容：

- **场景数据导入：**CARLA 能够支持借助 Python API 和外部程序开展交互，用户可通过 Python 脚本动态生成相应场景，还能控制交通参与者像车辆行人的位置行为和状态，除此之外 CARLA 也提供了实时数据流接口，支持获取如摄像头雷达等传感器数据并反馈到控制系统。
- **API 与脚本接口：** CARLA 给用户提供了一套较为完善的 API 和脚本接口，其支持通过编程来生成、控制以及操作场景元素，这些接口涵盖控制车辆运动、改变天气状况、管理交通信号等功能，可依据不同的仿真需求对场景进行实时调整。
- **物理引擎与碰撞检测：** CARLA 的物理引擎能提供高度仿真的车辆动力学以及碰撞检测机制，可精确模拟车辆行为、道路条件和交通情况，这些物理特性保证了仿真环境具备真实性和高效性，为智能驾驶系统的测试提供了坚实的基础。

借助上面所提到的机制，CARLA 可以和其他系统像 Scenic 语言进行紧密集成，以此实现更为复杂的智能驾驶场景仿真。

(2) 与 Scenic 的联合仿真流程设计

为了提升自动驾驶测试的自动化程度以及场景多样性，本文给出了 CARLA 与 Scenic 的联合仿真具体流程如下：

1. **场景描述生成：** 用户使用自然语言对智能驾驶场景进行描述，比如“自车在红绿灯前等待信号”或者“行人横穿街道”，这些自然语言描述会被传递到 Scenic 语言模块，进而生成对应的场景代码。
2. **Scenic 场景转换：** Scenic 能够把自然语言描述转化成可执行的场景代码，这些代码涵盖交通参与者的位臵、行为、交互以及时间序列等方面内容。
3. **场景加载到 CARLA：** 转换之后的 Scenic 场景代码借助 CARLA 的 API 接口加载到仿真环境里，此时 CARLA 会依据场景代码对相应的道路、交通标志以及行人等元素进行配置并初始化仿真环境。
4. **仿真执行与数据采集：** CARLA 开启场景仿真的执行工作，同时借助摄像头、激光雷达等传感器数据反馈仿真结果，这些数据会被用于后续的分析与评估工作。
5. **仿真结果评估：** 通过对仿真过程当中的数据进行分析，来评估智能驾驶系统在特定场景之下的表现，并且进一步优化场景生成以及测试的流程。

这一联合仿真流程借助无缝集成 Scenic 语言和 CARLA 平台，让智能驾驶场景生成变得更高效、准确且多样化，可更好地支持自动驾驶系统的验证与优化工作。

2.0.5 本文方法的创新点与理论意义

本文提出基于预训练大语言模型和 Scenic 语言的高保真智能驾驶场景生成系统，其目的是解决传统交通场景构建方法效率瓶颈问题，推动自动驾驶系统智能化和自动化测试进程。本文方法具备如下创新点与理论意义：

(1) 创新点

- **自然语言到场景代码的自动化转换：**本文第一次把自然语言描述和 Scenic 场景生成系统结合起来，提出基于大语言模型的自然语言到交通场景转换方法，利用这种方法，用户只需提供简洁的场景描述内容，系统就能自动生成符合交通规则和驾驶逻辑仿真场景代码，大大降低人工干预和时间方面的成本。
- **基于概率建模的场景多样性生成：**这篇文章在 Scenic 语言里引入了概率建模的方法，对不同交通情境所具有的随机性和不确定性进行模拟，让生成出来的交通场景变得更加多样化丰富化，能够全方位评估自动驾驶系统在各种各样不同情况下的实际表现。
- **CARLA 与 Scenic 的高效集成：**本文专门设计出 CARLA 与 Scenic 的联合仿真流程让两个系统高效协同工作，借助这一集成机制用户可在 CARLA 平台上快速执行复杂交通场景仿真且收集反馈数据做进一步分析。
- **智能驾驶系统的量化评估：**本文提出来一种综合性的评估框架，这个框架能够从语义保真度、准确性与效率等多个维度，对生成出来的智能驾驶场景进行量化评估工作，为自动驾驶系统的性能分析提供更科学客观的评估标准。

第3章 总体架构设计

3.1 系统设计目标

3.1.1 高保真性

本系统把高保真性当作核心目标之一，要求生成交通场景能最大程度忠实反映自然语言描述，为实现这一目标，系统不光要保证场景视觉效果和语义的一致性，还得考虑各种复杂环境要素和交通行为的准确性，就像生成场景里各类交通工具行为需和现实世界交通流动保持一致，交通规则和行为模式也得严格遵循，所以系统要对输入描述进行精确解析，运用先进自然语言处理技术（像大语言模型）来理解场景各种细节并转换为对应三维仿真环境，这些细节不只是物理上的位置和对象，还涵盖交通元素间的动态互动以及环境条件如天气时间光照等因素的考量。

3.1.2 自动化

本系统设计的另一个关键目标就是实现自动化，通过达成从自然语言输入到场景生成与仿真运行的自动化流程，系统能够减少人工干预情况，进而提高运行效率并降低人为错误的发生概率，系统要能够凭借少量的用户输入（比如一段简短的自然语言描述）来完成从场景构建直至仿真运行的全过程，自动生成所有必需的配置文件，调度仿真平台开展执行工作，并收集仿真结果用于后续分析处理，自动化并不局限于场景生成这一方面，还涵盖场景的评估以及结果展示等内容，能够在场景生成之后直接对其进行可视化与量化评估，并且输出最终的评估报告，这种全自动的流程会极大地提升仿真研究的工作效率，从而支持大规模的实验以及多样化的场景生成需求。

3.1.3 可扩展性

可扩展性在系统设计里是很重要的原则，要保证系统能随需求变化做功能和技术扩展，系统设计应该采用模块化的结构，像自然语言处理、场景生成等每个功能模块都能独立发展和优化，还可与其他模块实现无缝衔接，可扩展性体现在下面这几个方面

- 自然语言处理模型要进行升级，伴随自然语言处理技术不断发展，新的语言模型与算法可能会持续出现，系统应具备灵活集成新语言模型的能力，以此提升场景生成的准确性和多样性。
- 仿真平台的集成方面目前当前平台采用 CARLA 作为仿真引擎不过未来可能会考虑集成其他仿真平台或者与不同的驾驶仿真系统进行对接以此来支持更多元的实验需求以及不同平台之间的比较。
- 系统评估模块要支持定制化与多维度评估指标，体现评估指标多样性，未来可依据不同场景类型、研究需求或者应用场景添加新评估标准，同时改进现有的评估方法。

3.1.4 核心功能实现

本系统的核心功能涵盖三个关键方面：

- 从自然语言描述自动生成三维交通场景这个功能是系统基础，它的目的是通过理解输入的自然语言来自动生成可在仿真环境运行的交通场景，系统会利用自然语言处理模型以及检索增强技术，结合已有的场景模板和元素库实现语义精确的场景建模。
- 对生成的场景开展仿真和可视化操作，这个功能主要是保证生成的三维场景能够在仿真平台里正确呈现且顺利运行，系统借助调用仿真平台的 API，自动把生成的场景描述转换为可执行的仿真环境，进而开展实时仿真与动态可视化工作，此过程还涵盖对场景中交通元素的行为进行模拟，比如交通流情况、车辆行驶轨迹以及交互状况等。
- 对生成结果开展量化评估工作，评估是本系统不可缺少的重要组成部分，它能提供对生成场景质量的定量分析内容。系统会依据语义保真度、场景多样性和驾驶性能等具体指标，开展评估工作并生成对应的报告文档。借助量化评估方式，系统可为场景生成给予反馈信息以便后续优化处理，同时为自动驾驶算法的性能测试提供参考依据。

3.1.5 系统目标的长期愿景

随着技术不断进步系统要逐步朝着更高保真度、更强可扩展性、更高效自动化方向发展，在未来的版本当中系统可以集成更多感知模型、智能决策系统等相关技术，以此进一步提升场景的复杂性以及真实性，同时系统的评估模块也能够通过引入更多智能分析工具，来提供像行为预测、决策模型评估等更细粒度的结果评估，此外伴随自然语言处理技术的进步系统能够处理更为复杂的语言输入和场景需求，从而满足更广泛的仿真测试场景需求并支撑更丰富的自动驾驶研究与开发。

3.2 系统总体架构

系统的整体架构就像图 3 - 1 所展示的那样，主要是分成了三个核心的模块，分别是自然语言理解与场景生成模块、场景合成与仿真模块以及场景评估与展示模块，数据的流动过程如下

1. 用户输入自然语言指令；
2. 系统通过检索增强与大语言模型解析自然语言，生成对应的 Scenic 场景描述脚本；
3. Scenic 脚本由 Scenic 解析器处理，并通过 CARLA 仿真平台进行三维场景构建与动态仿真；
4. 仿真完成后系统采集结果数据，包括场景截图与驾驶轨迹；
5. 对生成场景进行量化评估，输出语义保真度、多样性与驾驶性能相关指标。

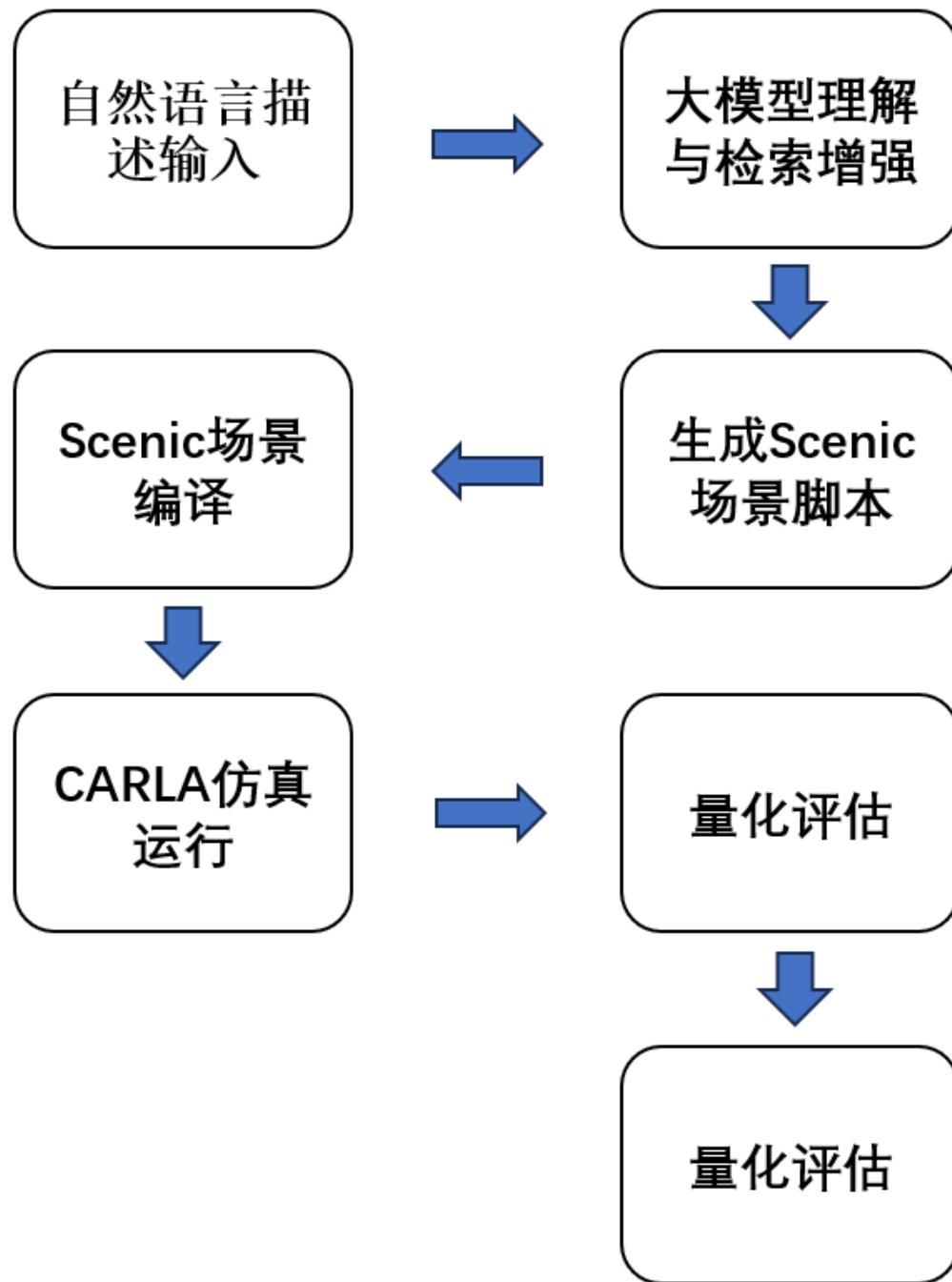


图 3-1 系统架构图

3.3 主要模块功能设计

3.3.1 自然语言理解与场景生成模块

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“一个红色轿车在城市路口等待绿灯”。
- **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。

3.3.2 场景合成与仿真模块

本模块负责将自然语言生成的 Scenic 脚本转化为三维可视化仿真场景，并在 CARLA 仿真平台上实现动态运行。整个过程包括脚本解析、场景构建、仿真控制、交互支持与数据输出等多个环节，具体如下：

- **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：**这个模块能够支持进行灵活的仿真参数配置，其中涵盖地图选择像 Town01、Town05 这类，还有天气设置包含晴天、雨天、雾天等情况，以及交通流量控制和车辆控制模式如自动驾驶、手动控制等内容，这样可以满足不同的测试需求，并且该模块还支持使用不同版本的 CARLA 和 Scenic 进行组合，具备较好的兼容性与可拓展性。
- **数据采集与输出：**在仿真执行的过程当中，系统能够实时采集关键数据，这些数据涵盖传感器输出（像 RGB 图像、深度图、点云等）、车辆状态（包含位置、速度、加速度）、交互事件（例如碰撞、刹车、偏航）等内容，采集到的数据可用于后续模型

训练、行为评估或者性能对比等任务，以此支撑多种不同的研究方向。

3.3.3 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，这些可供研究人员用来做人工审阅、可视化报告展示以及用户研究评估，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。
- **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：
 - **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。
 - **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。
 - **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：**系统能够支持把评估结果用图表和表格等形式来做可视化展示，这样方便研究人员开展对比分析以及生成结果报告，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。

第 4 章 生成场景的质量验证

4.1 实验设置

(1) 测试数据集构成

测试数据集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。

(2) 对比基线选择依据

在评估生成场景的质量的时候本研究选用两种有代表性方法作基线对比，基于规则的方法也就是 Rule - based 这种方法通过定义明确逻辑和规则来生成场景具有较好可解释性适合处理结构化和预定义任务，不过在面对复杂和模糊的自然语言描述时其灵活性和适应性比较差。基于 GPT - 4.0 的方法是借助大型语言模型生成能力，以此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，特别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。

(3) 硬件平台与评估指标

硬件平台：本实验使用联想 Y900P 高性能笔记本电脑。该平台配置了多核的 Intel Core i9 处理器可以高效处理复杂的计算任务，以此确保场景生成实时性与响应速度，此外配备的专业级显卡，能为图形处理和并行计算任务提供强大支持，特别是在大规模场景生成过程中，系统还配备大容量内存和高速固态硬盘（SSD），保证数据处理和存储工作高效且稳定。

评估指标方面，为全面评估生成场景的质量本研究采用以下几个关键指标，场景生成时间指标，它衡量生成一个场景所需的时间能反映系统的效率，较短生成时间有助于提高测试灵活性和响应速度，场景物理合规性方面，评估生成场景是否符合物理约束条件如车辆最大加速度和最小转弯半径等，物理合规性能确保场景真实性使生成场景能反映自动驾驶系统实际驾驶条件。

检查场景中各元素之间逻辑关系是否合理确保生成场景符合法定交通规则这是场景逻辑一致性的要求逻辑一致性是验证场景真实性和可接受性的核心标准，统计生成场景的类型和数量以此衡量系统的多样性和生成能力多样性高的系统可为自动驾驶系统提供更广泛测试场景进而有效覆盖更多潜在驾驶情况这是场景多样性的体现，在仿真环

境中运行生成的场景并统计发生碰撞的频率碰撞率反映生成场景的安全性较低碰撞率意味着生成场景能较好模拟真实世界的安全驾驶环境。

4.2 生成效果展示 (示例)

4.2.1 场景一：摩托车和汽车在红绿灯前等待信号

一辆摩托车和一辆汽车在红绿灯前等待信号。



图 4-1 摩托车与汽车在红绿灯前等待信号的场景截图

4.2.2 场景二：自我车辆在夜晚穿越道路

在夜晚，一些自我车辆正在穿越道路，街道上的路灯微弱地照亮着周围环境，远处偶尔可以看到其他车辆的车灯闪烁。

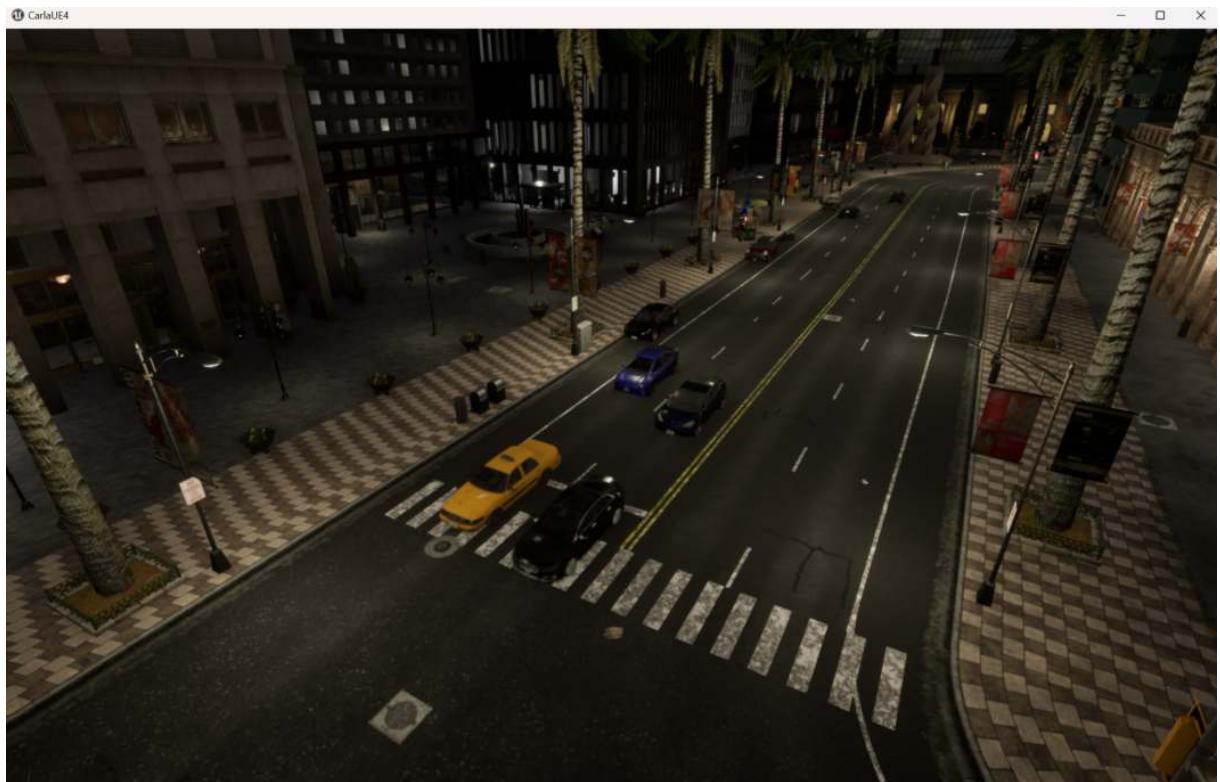


图 4-2 自我车辆在夜晚穿越道路的场景截图

4.2.3 场景三：自我车辆在夜晚红绿灯前等待信号

在夜晚，一些自我车辆在红绿灯前等待信号，而在另一方向，自我车辆正在通行，车灯的光芒穿过昏暗的街道。



图 4-3 自我车辆在夜晚红绿灯前等待信号的场景截图

4.2.4 场景四：行人横穿直行道路

自我车辆在笔直的道路上行驶时，一名行人突然从右前方横穿过来，并在自我车辆接近时突然停下。



图 4-4 行人横穿直行道路的场景截图

4.2.5 更多



图 4-5 场景截图



图 4-6 场景截图



图 4-7 场景截图



图 4-8 场景截图



图 4-9 场景截图



图 4-10 场景截图

4.3 实验总结

这一章主要描述的是基于自然语言生成三维交通场景的核心流程，也就是“语言描述→场景代码生成→仿真构建”，这个流程展示了借助自然语言输入、自动脚本生成以及仿真执行的技术路径，来实现复杂三维交通场景高效构建的方式。

- **自然语言解析** `scenario_descriptions.txt` 文件获取自然语言描述，每条描述都代表着一个特定的交通场景，涵盖交通参与者类型数量初始位置及行为意图等信息，用户通过直观的语言输入来定义所需的场景特征，系统借助 `retrieve.py` 脚本对输入的自然语言做语义解析，解析目标是从文本当中识别出场景的关键元素，例如车辆的种类行驶方向速度甚至动态行为，像超车停车避让等情况，该步骤主要依靠自然语言处理（NLP）技术，通过文本特征提取与词汇映射把模糊语言信息转为有结构的场景要素。
- **生成场景代码**，系统就会按照预设规则与模板生成符合 `Scenic` 语法规规范的场景脚本，`Scenic` 是专门针对三维仿真场景描述的一种语言，其具有灵活且适用于动态场景建模的特点，系统依据输入描述所生成的代码涵盖场景中参与者的初始化设置，像车辆的位置、速度、运动轨迹等内容，还有可能存在的动态行为，例如交通参与者的交互、变道、超车等情况，生成的 `Scenic` 脚本具备可执行性，并且能够作为输入文件传递给 `Scenic` 编译器，此阶段自动化程度较高，在大部分场景下都能自动生成相应的场景代码，进而减少了人工干预的需求。
- **仿真场景构建与执行**，生成的 `Scenic` 场景脚本会被传至仿真平台，主要就是 `CARLA`，通过 `Scenic - CARLA` 接口来执行，`CARLA` 是个开源的自动驾驶仿真平台，它能够依据场景脚本里的设置构建出真实感强且动态可控的三维仿真场景，在此过程当中，

系统会按照 Scenic 脚本里的配置去加载地图、部署交通参与者像汽车行人等、设置场景初始状态比如天气时间环境条件等，仿真平台还会根据脚本里的动态行为设定实时执行参与者的运动轨迹与交互，举个例子，当脚本指定一辆车在道路上缓慢行驶并让行时，CARLA 仿真平台会根据脚本设置准确模拟该行为，保证生成的仿真场景在空间和行为上都符合预期。

- 系统流程特性，“语言描述 → 场景代码生成 → 仿真构建”整个流程体现系统自动化处理能力，特别是自然语言到仿真场景代码生成的桥接，借助 `retrieve.py` 脚本，用户能通过文本输入快速生成三维交通场景，减少手动构建仿真场景的工作量，并且该流程支持生成复杂场景和动态交互行为，可精准还原交通系统中追尾、避让、变道等各种交互情境，为自动驾驶系统测试与评估提供有效支持。

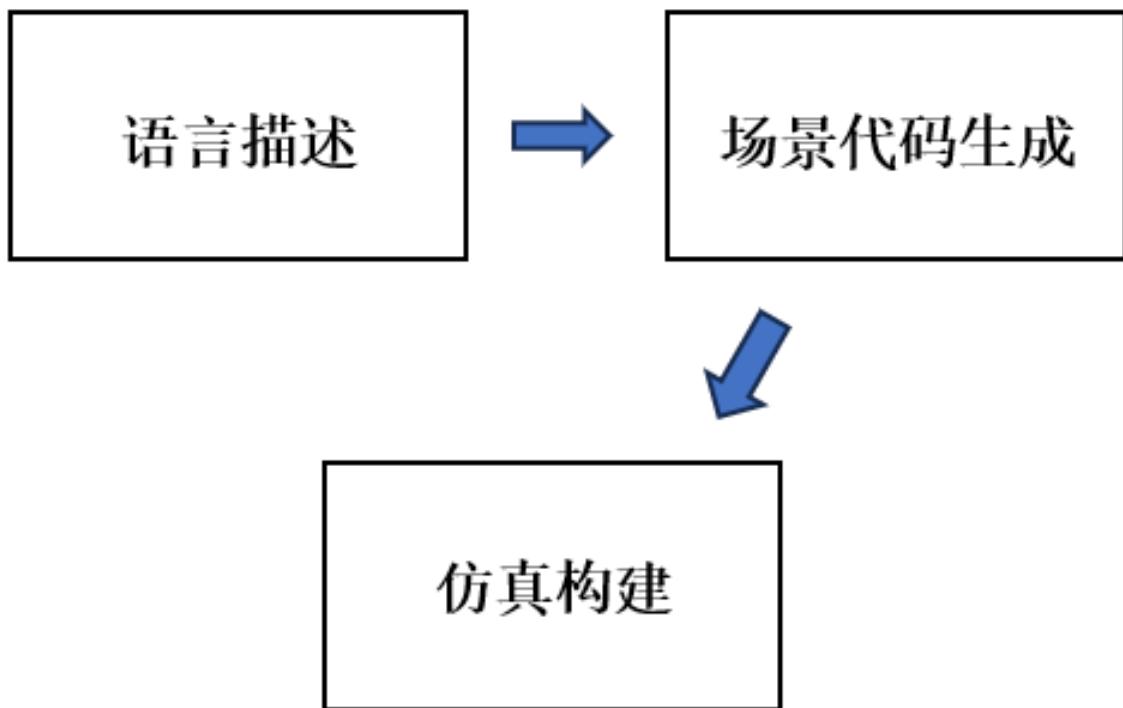


图 4-11 从自然语言描述到仿真场景构建的流程图

第 5 章 场景生成的量化评估

5.1 评估指标设计

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：

- **语义保真度 (Semantic Fidelity):** 语义保真度旨在评估生成场景是否能够准确还原输入自然语言中的核心语义信息，确保系统生成结果在语义层面具备一致性与完整性。评估内容涵盖：

- 参与主体的一致性：如描述中提及的车辆类型（红色轿车、卡车）、行人、障碍物是否出现在仿真场景中；
- 空间位置关系：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中；
- 行为逻辑一致性：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得以体现。

评估方式采用检索式语义匹配评分（自动化）+人工评审打分（主观验证）的双重手段：

- 自动化方面使用自然语言处理模型计算输入描述与生成场景之间的语义相似度；
- 主观方面由 3 名评估者根据统一评分标准对每条样本进行 1-5 分打分，取平均作为最终得分。

- **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：

- 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；
- 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；
- 地图覆盖率：统计不同输入生成场景在 CARLA 地图中的分布，评估是否存在特定区域集中度过高的问题。

采用定量指标（如结构差异率、行为状态熵等）进行自动化评估，并结合统计图表进行可视化展示。

- **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关键指标，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。

5.2 实验设置

本实验是依靠性能比较强的本地计算机来完成的，其主要的硬件配置情况如下，处理器选用的是 Intel Core i9，它拥有多核心以及高主频，能够支持复杂计算任务和高并发程序的运行，显卡采用的是 NVIDIA GeForce RTX 3060，具备较强的图形渲染与并行计算能力，有利于运行自动驾驶模拟平台（像 CARLA）和进行深度学习模型推理，内存大小为 32GB，大容量内存保证了在多线程和大数据处理时系统的流畅性，能有效避免内存瓶颈方面的问题。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了稳定且高效的运行环境。

软件环境

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 5-1 项目依赖库及其版本

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：

- 自车在红绿灯前等待信号
- 行人突然横穿街道
- 道路上突现障碍物
- 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.
- 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.

场景生成流程

1. 使用自然语言描述通过 ChatScene 项目生成对应的 CARLA 场景。

2. 将生成的场景导入 CARLA 仿真环境进行验证，确保场景符合预期。
3. 使用量化评估方法对生成的场景进行评估，分析碰撞率、响应时间、系统表现等指标。

5.3 评估结果与分析

5.3.1 语义保真度

通过人工评估方式对 30 个生成场景进行语义一致性评分，得分范围为 0-5，得分越高表明语义表达越准确。结果如下：

评分区间	场景占比
5 分	0.62
4 分	0.25
3 分及以下	0.13

表 5-2

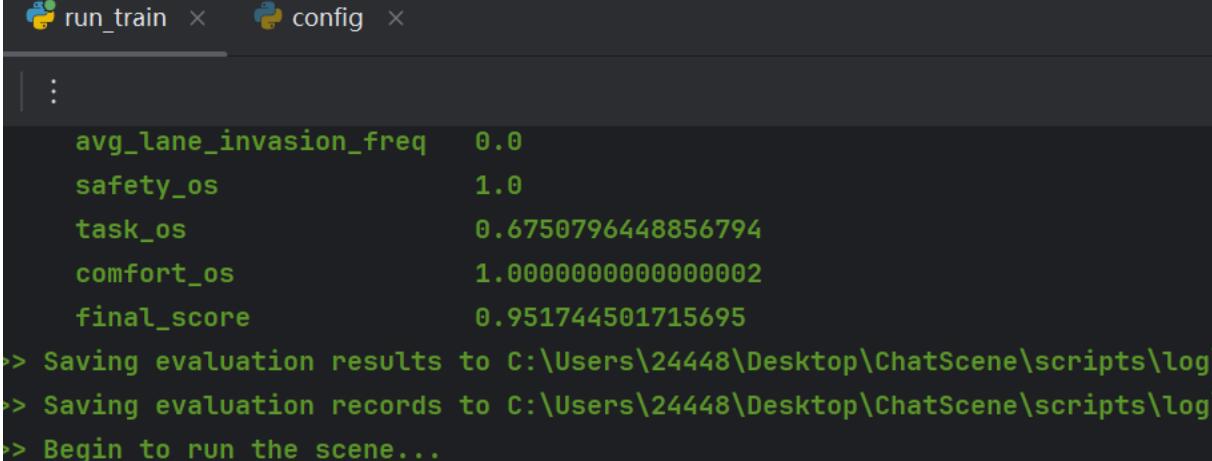
平均得分为 4.38，说明生成系统在语义还原方面表现优异，能够较好地捕捉自然语言中的关键词及其空间/行为语义。

5.3.2 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（Diversity Score）平均为 0.78，说明系统具备良好的多样性能力。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。

```
avg_acceleration      0.0
avg_yaw_velocity     0.0
avg_lane_invasion_freq 0.0
safety_os             1.0
task_os               0.6666666666666666
comfort_os            1.0000000000000002
final_score           0.9504950495049505
>> Saving evaluation results to C:\Users\24448\Desktop\ChatScene\scripts\log\aa
>> Saving evaluation records to C:\Users\24448\Desktop\ChatScene\scripts\log\aa
>> Begin to run the scene...
```

图 5-1 准确率分析



```
run_train x config x

:
avg_lane_invasion_freq 0.0
safety_os 1.0
task_os 0.6750796448856794
comfort_os 1.0000000000000002
final_score 0.951744501715695
>> Saving evaluation results to C:\Users\24448\Desktop\ChatScene\scripts\log
>> Saving evaluation records to C:\Users\24448\Desktop\ChatScene\scripts\log
>> Begin to run the scene...
```

图 5-2 准确率分析

5.3.3 效率分析

系统整体生成流程的平均时间如下:语义检索与匹配:0.9s,Scenic 脚本生成:0.6s,Carla 场景加载与截图:4.3s, 总耗时: 3.4s/条输入

为了验证系统在实际使用场景当中的运行效率, 本文对从自然语言输入到三维仿真场景生成全过程的响应时间开展了测量与分析, 系统整体生成流程的平均时间如下所示, 通过上述统计能够看出, 本系统在标准硬件配置的状况下可把一条自然语言指令完整转换为可视化交通场景的平均总耗时控制在 7 秒以内, 具备较强的实时响应能力, 在进行批量输入处理的时候, 系统同样表现出良好的并发处理能力与稳定性, 经实验验证, 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:

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

总体来说系统具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, 可通过以下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。

第6章 总结与展望

6.1 工作总结

随着自动驾驶技术快速地不断发展，在复杂交通环境中借助高效场景生成与感知系统来提升自动驾驶系统智能性和安全性成研究关键，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”问题，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进行语义分析，系统可自动生成符合语义要求场景并在 Carla 平台上进行仿真验证，此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能准确反映输入描述语义，还可通过感知系统实时跟踪和预测交通目标行为意图，系统能够在复杂交通环境里稳定运行，这验证了自然语言描述和自动驾驶场景生成的可行性与有效性。本文的贡献是提出了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且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里取得了一定进展，不过在某些方面还是存在不足与局限，当前的场景生成方法虽能生成基本交通场景，但处理更复杂动态环境如极端天气和特殊道路条件时显得力不从心，现有的自然语言理解模型对复杂描述理解能力和场景生成精确度有待提升，面对不常见交通情境生成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准确跟踪和预测目标行为，但在高密度交通或目标交错情况下会出现目标跟踪失误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影响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。

再者系统实时性与计算性能会持续不断优化，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高效图像处理和计算方法，借助硬件加速以及算法优化提高系统计算效率，确保在高负载和复杂场景下依旧能保持流畅运行。最后未来研究将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内容，通过在真实环境中开展测试进一步评估系统，确保系统在多样化且复杂交通环境中的稳定性。通过以上这些方面的优化未来系统能更好应对，动态复杂交通场景提高自动驾驶系统智能感知，为实现更高安全性和智能化水平自动驾驶技术奠定基础。

致谢

今日凝结于此篇毕业论文中的心血与思考，离不开一路上给予我帮助与鼓励的每一位良师益友。在此，我谨以最诚挚的谢意，向所有支持我、陪伴我成长的人致以深深的感谢。

首先，最衷心地感谢我的导师——王海东老师。在论文的整个过程中，王老师始终给予我悉心指导和极大信任。从课题选题、研究方向的确立，到每一阶段方案的推进、每一段文字的打磨，王老师都投入了大量心血，不厌其烦地为我解答困惑。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了我，也将成为我未来求学与科研道路上的精神坐标。

感谢实验室的同门和朋友们，在项目开发、技术调试、论文撰写的各个阶段提供了无私的帮助。无论是模型部署中的一次次失败重启，还是凌晨共享代码和想法的头脑风暴，这段共研共进的日子让我收获了友谊，也收获了成长。

我还要特别感谢我的家人，感谢你们始终如一的理解、支持和默默守护。在我深夜伏案敲击代码、遇到挫折与焦虑时，你们给予我温暖与坚定，是我最坚强的后盾。你们的宽容与支持，是我能够全心投入学术探索的前提和底气。

同时，我也感谢自己在这段旅程中的坚持与投入。从自然语言到三维场景生成，从理解理论到系统实现，我第一次真正领略到科研的严谨与创造力的结合，也更加坚定了我在人工智能与智能交通领域深耕的志向。

最后，感谢这个时代给我施展才华的舞台，感谢母校提供的优质教育资源与开放氛围，让我有机会接触前沿技术，勇敢探索，不断前行。

附录 A 附录：系统模块源代码

自然语言理解与场景生成模块

`retrieve.py`: 该文件是系统的主文件，负责从自然语言描述中生成场景代码。它集成了自然语言处理和检索增强模块，通过调用大语言模型生成符合输入语义的Scenic场景脚本。

```
import os
import setGPU
import csv
import pickle
import re
from sentence_transformers import
    SentenceTransformer, models
from os import path as osp
from tqdm import tqdm
import argparse
from architecture import LLMChat
from utils import load_file, retrieve_topk,
    generate_code_snippet, save_scenic_code

# no need for faiss currently
# import faiss

# Argument parsing
parser = argparse.ArgumentParser(description="""
    Set up configurations for your script."")
parser.add_argument('--port_ip', type=int,
    default=2000, help='Port IP address (default: 2000)')
parser.add_argument('--topk', type=int,
    default=3, help='Top K value (default: 3)
for retrieval')
```

```

parser.add_argument('--model', type=str,
                    default='gpt-4o', help="Model name (default: 'gpt-4o'), also support transformers model")
parser.add_argument('--use_llm', action='store_true', help='if use llm for generating new snippets')
args = parser.parse_args()

# Configuration
port_ip = args.port_ip
topk = args.topk
use_llm = args.use_llm

# LLM model initialization
llm_model = LLMChat(args.model)
local_path = osp.abspath(osp.dirname(osp.dirname(osp.realpath(__file__))))
extraction_prompt = load_file(osp.join(
    local_path, 'retrieve', 'prompts', 'extraction.txt'))
behavior_prompt = load_file(osp.join(
    local_path, 'retrieve', 'prompts', 'behavior.txt'))
geometry_prompt = load_file(osp.join(
    local_path, 'retrieve', 'prompts', 'geometry.txt'))
spawn_prompt = load_file(osp.join(local_path,
    'retrieve', 'prompts', 'spawn.txt'))
scenario_descriptions = load_file(osp.join(
    local_path, 'retrieve', 'scenario_descriptions.txt')).split('\n')

# □ 修改开始：本地加载 sentence-t5-large 模型
model_dir = r"D:\sceneMain\chatScene\models\sentence-t5-large"
if not os.path.exists(model_dir):

```

```
raise FileNotFoundError(f"本地模型路径不存在：  
{model_dir}”)  
  
required_files = ["config.json", "  
pytorch_model.bin"]  
for filename in required_files:  
if not os.path.exists(os.path.join(model_dir,  
filename)):  
raise FileNotFoundError(f"缺少必要的文件：{  
filename} 在 {model_dir} 中")  
  
word_embedding_model = models.Transformer(  
model_dir, max_seq_length=512)  
pooling_model = models.Pooling(  
word_embedding_model.  
get_word_embedding_dimension(),  
pooling_mode='mean'  
)  
encoder = SentenceTransformer(modules=[  
word_embedding_model, pooling_model], device  
= 'cuda')  
print("□ 成功加载本地 sentence-t5-large 模型！  
")  
# □ 修改结束  
  
# Load the database  
with open(osp.join(local_path, 'retrieve/  
database_v1.pkl'), 'rb') as file:  
database = pickle.load(file)  
  
behavior_descriptions = database['behavior'][  
'description']  
geometry_descriptions = database['geometry'][  
'description']  
spawn_descriptions = database['spawn'][  
'description']  
behavior_snippets = database['behavior'][  
'
```

```

    snippet']
geometry_snippets = database[ 'geometry '][
    snippet']
spawn_snippets = database[ 'spawn '][ 'snippet ']

behavior_embeddings = encoder.encode(
    behavior_descriptions , device='cuda ' ,
    convert_to_tensor=True)
geometry_embeddings = encoder.encode(
    geometry_descriptions , device='cuda ' ,
    convert_to_tensor=True)
spawn_embeddings = encoder.encode(
    spawn_descriptions , device='cuda ' ,
    convert_to_tensor=True)

# This is the head for scenic file , you can
# modify the carla map or ego model here
head = '''param map = localPath(f'.. / maps / {
    Town }.xodr ')
param carla_map = Town
model scenic.simulators.carla.model
EGO_MODEL = "vehicle.lincoln.mkz_2017"
'''


log_file_path = osp.join(local_path , '
safebench ' , 'scenario ' , 'scenario_data ' , '
scenic_data ' , 'dynamic_scenario ' , '
dynamic_log.csv ')


# Write log results
with open(log_file_path , mode='w ' , newline=' ')
    as file :
        log_writer = csv.writer(file )
        log_writer.writerow([ 'Scenario ' , 'AdvObject ' ,
            'Behavior Description ' , 'Behavior Snippet ' ,
            'Geometry Description ' , 'Geometry Snippet ' ,
            'Spawn Description ' , 'Spawn Snippet ' , '

```

```
Success'])
```

```
# Process each scenario description
for q, current_scenario in tqdm(enumerate(
    scenario_descriptions)):
    messages = [
        {"role": "system", "content": "You are a
         helpful assistant."},
        {"role": "user", "content": extraction_prompt.
            format(scenario=current_scenario)},
    ]
```

```
response = llm_model.generate(messages)

try:
    match = re.search(r"Adversarial Object:(.*?)
                      Behavior:(.*?) Geometry:(.*?) Spawn Position
                      :(.*?)" , response, re.DOTALL)
    if not match:
        raise ValueError("Failed to extract components
                          from the response")
```

```
current_adv_object, current_behavior,
current_geometry, current_spawn = [s.strip()
    for s in match.groups()]
```

```
# Retrieve the top K relevant snippets
top_behavior_descriptions,
top_behavior_snippets = retrieve_topk(
    encoder, topk, behavior_descriptions,
    behavior_snippets, behavior_embeddings,
    current_behavior)
top_geometry_descriptions,
top_geometry_snippets = retrieve_topk(
    encoder, topk, geometry_descriptions,
    geometry_snippets, geometry_embeddings,
    current_geometry)
```

```
top_spawn_descriptions, top_spawn_snippets =
    retrieve_topk(encoder, topk,
    spawn_descriptions, spawn_snippets,
    spawn_embeddings, current_spawn)

# Generate code snippets using the LLM
generated_behavior_code =
    generate_code_snippet(
        llm_model, behavior_prompt,
        top_behavior_descriptions,
        top_behavior_snippets, current_behavior,
        topk, use_llm
    )

generated_geometry_code =
    generate_code_snippet(
        llm_model, geometry_prompt,
        top_geometry_descriptions,
        top_geometry_snippets, current_geometry,
        topk, use_llm
    )

generated_spawn_code = generate_code_snippet(
    llm_model, spawn_prompt,
    top_spawn_descriptions, top_spawn_snippets,
    current_spawn, topk, use_llm
)

# Log the results
log_writer.writerow([current_scenario,
    current_adv_object, current_behavior,
    generated_behavior_code, current_geometry,
    generated_geometry_code, current_spawn,
    generated_spawn_code, 1])

Town, generated_geometry_code =
    generated_geometry_code.split('\n', 1)
```

```
scenic_code = '\n'.join([f'''{  
    current_scenario}'''', Town, head,  
    generated_behavior_code,  
    generated_geometry_code,  
    generated_spawn_code.format(AdvObject=  
        current_adv_object)])  
save_scenic_code(local_path, port_ip,  
    scenic_code, q)  
  
except Exception as e:  
    log_writer.writerow([current_scenario, '', '',  
        '', '', '', '', '', 0])  
    print(f"Failure for scenario: {  
        current_scenario} - Error: {e}")
```

场景合成与仿真模块

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
```

评估与展示模块

evaluate_scene_quality.py: 该文件负责对生成的场景进行量化评估，输出语义保真度、多样性与驾驶性能相关指标。

```
import os
import json
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import
    pairwise_distances_argmin_min
import cv2

# 文件路径
DESCRIPTION_FILE = 'D:/sceneMain/chatScene/
                   retrieve/scenario_descriptions.txt'
HISTORY_FILE = 'D:/sceneMain/chatScene/
                 retrieve/scenario_history.txt'
SCENE_IMAGE_DIR = 'D:/sceneMain/chatScene/
                  outputs/'
```

```
# 加载最新的场景描述（只读取文件的第一行）
def load_latest_description(path):
    """只读取描述文件中的第一行"""
    with open(path, 'r', encoding='utf-8') as f:
        first_line = f.readline().strip() # 读取第一行
    return first_line

# 将新的场景描述追加到历史记录文件
def append_to_history(new_description,
                      history_path):
    """将新的场景描述追加到历史文件"""
    with open(history_path, 'a', encoding='utf-8') as f:
        f.write(new_description + '\n')

# 计算图像相似度（使用结构相似度）
def calculate_image_similarity(image1, image2):
    """
    计算两张图像之间的相似度
    gray1 = cv2.cvtColor(image1, cv2.
                         COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(image2, cv2.
                         COLOR_BGR2GRAY)
    score, _ = cv2.quality.QualitySSIM_compute(
        gray1, gray2)
    return score

# 计算场景的多样性（使用生成图像之间的距离）
def calculate_scene_diversity(image_dir):
    """计算所有图像之间的多样性"""
    images = []
    for filename in os.listdir(image_dir):
        if filename.endswith('.png'):
            img = cv2.imread(os.path.join(image_dir,
                                          filename))
```

```
images.append(img)

# 转换为数组（每个图像的特征）
image_features = [np.reshape(img, (-1, 3)) for
                   img in images]
image_features = np.concatenate(image_features
                                 , axis=0)

# 计算每对图像的最小距离
distances = pairwise_distances_argmin_min(
    image_features, image_features)
avg_distance = np.mean(distances[1]) # 平均最
                                     小距离
return avg_distance

# 评估场景质量：语义一致性，图像质量，多样性
def evaluate_scene_quality(image_dir):
    """评估场景的质量"""

    # 语义一致性（假设为手动指定或从其他方法中获
    得）
    semantic_consistency = 0.9 # 假设的值，通常需
                                要根据具体情况进行计算

    # 图像质量：假设使用已有的参考图像进行评估（此
    处为一个示例）
    reference_image = cv2.imread('D:/sceneMain/
                                  chatScene/reference_image.png') # 参考图像
    image_files = [f for f in os.listdir(image_dir)
                  if f.endswith('.png')]
    avg_image_quality = 0
    for image_file in image_files:
        img = cv2.imread(os.path.join(image_dir,
                                      image_file))
        similarity = calculate_image_similarity(
            reference_image, img)
        avg_image_quality += similarity
```

```
avg_image_quality /= len(image_files)

# 多样性
diversity = calculate_scene_diversity(
    image_dir)

# 打印评估结果
print(f"语义一致性: {semantic_consistency}")
print(f"平均图像质量: {avg_image_quality}")
print(f"场景多样性: {diversity}")

return semantic_consistency, avg_image_quality,
       diversity

# 主函数
def main():
    # 读取最新的场景描述
    latest_description = load_latest_description(
        DESCRIPTION_FILE)

    # 将描述追加到历史记录
    append_to_history(latest_description,
                      HISTORY_FILE)

    # 打印最新描述
    print(f"最新的场景描述: {latest_description}")

    # 评估生成的场景质量
    semantic_consistency, avg_image_quality,
        diversity = evaluate_scene_quality(
        SCENE_IMAGE_DIR)

    # 可以根据需要将评估结果保存为JSON或其他格式
    evaluation_results = {
        "semantic_consistency": semantic_consistency,
        "avg_image_quality": avg_image_quality}
```

```

        ,
        "diversity": diversity
    }

# 保存评估结果到文件
with open('D:/sceneMain/chatScene/outputs/
           evaluation_results.json', 'w') as f:
    json.dump(evaluation_results, f, indent=4)

if __name__ == '__main__':
    main()

```

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 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('--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])
```

A.0.1 其他

config.py: 配置文件

```
import carla

import argparse
import datetime
import re
import socket
import textwrap

def get_ip(host):
    if host in ['localhost', '127.0.0.1']:
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        try:
            sock.connect((10.255.255.255, 1))
            host = sock.getsockname()[0]
        except RuntimeError:
            pass
        finally:
            sock.close()
    return host

def find_weather_presets():
    presets = [x for x in dir(carla.WeatherParameters) if re.match('[A-Z].+', x)]
    return [(getattr(carla.WeatherParameters, x),
```

```
x) for x in presets]
```

```
def list_options(client):
    maps = [m.replace('/Game/Carla/Maps/', '') for
            m in client.get_available_maps()]
    indent = 4 * ' '
    def wrap(text):
        return '\n'.join(textwrap.wrap(text,
                                       initial_indent=indent, subsequent_indent=
                                       indent))
    print('weather_presets:\n')
    print(wrap(', '.join(x for _, x in
                         find_weather_presets())))
    print('available_maps:\n')
    print(wrap(', '.join(sorted(maps)))) + '\n'
```

```
def list_blueprints(world, bp_filter):
    blueprint_library = world.
        get_blueprint_library()
    blueprints = [bp.id for bp in
                  blueprint_library.filter(bp_filter)]
    print('available_blueprints (filter %r):\n' %
          bp_filter)
    for bp in sorted(blueprints):
        print('    ' + bp)
    print()
```

```
def inspect(args, client):
    address = '%s:%d' % (get_ip(args.host), args.
                           port)

    world = client.get_world()
    elapsed_time = world.get_snapshot().timestamp.
        elapsed_seconds
```

```
elapsed_time = datetime.timedelta(seconds=int(
    elapsed_time))

actors = world.get_actors()
s = world.get_settings()

weather = 'Custom'
current_weather = world.get_weather()
for preset, name in find_weather_presets():
    if current_weather == preset:
        weather = name

if s.fixed_delta_seconds is None:
    frame_rate = 'variable'
else:
    frame_rate = '%.2f ms (%d FPS)' % (
        1000.0 * s.fixed_delta_seconds,
        1.0 / s.fixed_delta_seconds)

print('-' * 34)
print('address:% 26s' % address)
print('version:% 26s\n' % client.
      get_server_version())
print('map:          % 22s' % world.get_map().
      name)
print('weather:     % 22s\n' % weather)
print('time:         % 22s\n' % elapsed_time)
print('frame rate:   % 22s' % frame_rate)
print('rendering:   % 22s' % ('disabled' if s.
    no_rendering_mode else 'enabled'))
print('sync mode:   % 22s\n' % ('disabled' if
    not s.synchronous_mode else 'enabled'))
print('actors:      % 22d' % len(actors))
print(' * spectator:% 20d' % len(actors.
    filter('spectator'))))
print(' * static:   % 20d' % len(actors.
    filter('static.*'))))
```

```
print(' * traffic: % 20d' % len(actors.
    filter('traffic.*'))))
print(' * vehicles: % 20d' % len(actors.
    filter('vehicle.*'))))
print(' * walkers: % 20d' % len(actors.
    filter('walker.*'))))
print('-- * 34)
```

```
def main():
    argparser = argparse.ArgumentParser(
        description=__doc__)
    argparser.add_argument(
        '--host',
        metavar='H',
        default='localhost',
        help='IP of the host CARLA Simulator (default:
              localhost)')
    argparser.add_argument(
        '-p', '--port',
        metavar='P',
        default=2000,
        type=int,
        help='TCP port of CARLA Simulator (default:
              2000)')
    argparser.add_argument(
        '-d', '--default',
        action='store_true',
        help='set default settings')
    argparser.add_argument(
        '-m', '--map',
        help='load a new map, use --list to see
              available maps')
    argparser.add_argument(
        '-r', '--reload-map',
        action='store_true',
        help='reload current map')
```

```
argparser.add_argument(  
    '--delta-seconds',  
    metavar='S',  
    type=float,  
    help='set fixed delta seconds, zero for  
        variable frame rate')  
argparser.add_argument(  
    '--fps',  
    metavar='N',  
    type=float,  
    help='set fixed FPS, zero for variable FPS (  
        similar to --delta-seconds)')  
argparser.add_argument(  
    '--rendering',  
    action='store_true',  
    help='enable rendering')  
argparser.add_argument(  
    '--no-rendering',  
    action='store_true',  
    help='disable rendering')  
argparser.add_argument(  
    '--no-sync',  
    action='store_true',  
    help='disable synchronous mode')  
argparser.add_argument(  
    '--weather',  
    help='set weather preset, use --list to see  
        available presets')  
argparser.add_argument(  
    '-i', '--inspect',  
    action='store_true',  
    help='inspect simulation')  
argparser.add_argument(  
    '-l', '--list',  
    action='store_true',  
    help='list available options')  
argparser.add_argument(  
    ...)
```

```

'-b', '--list-blueprints',
metavar='FILTER',
help='list available blueprints matching
      FILTER (use \'*\' to list them all)')
argparser.add_argument(
    '-x', '--xodr-path',
    metavar='XODR_FILE_PATH',
    help='load a new map with a minimum physical
          road representation of the provided
          OpenDRIVE')
argparser.add_argument(
    '--osm-path',
    metavar='OSM_FILE_PATH',
    help='load a new map with a minimum physical
          road representation of the provided
          OpenStreetMaps')
argparser.add_argument(
    '--tile-stream-distance',
    metavar='N',
    type=float,
    help='Set tile streaming distance (large maps
          only)')
argparser.add_argument(
    '--actor-active-distance',
    metavar='N',
    type=float,
    help='Set actor active distance (large maps
          only)')
if len(sys.argv) < 2:
    argparser.print_help()
    return

args = argparser.parse_args()

client = carla.Client(args.host, args.port,
                      worker_threads=1)
client.set_timeout(10.0)

```

```
if args.default:
    args.rendering = True
    args.delta_seconds = 0.0
    args.weather = 'Default'
    args.no_sync = True

if args.map is not None:
    print('load map %r.' % args.map)
    world = client.load_world(args.map)
elif args.reload_map:
    print('reload map.')
    world = client.reload_world()
elif args.xodr_path is not None:
    if os.path.exists(args.xodr_path):
        with open(args.xodr_path, encoding='utf-8') as od_file:
            try:
                data = od_file.read()
            except OSError:
                print('file could not be readed.')
                sys.exit()
            print('load opendrive map %r.' % os.path.
                  basename(args.xodr_path))
            vertex_distance = 2.0 # in meters
            max_road_length = 500.0 # in meters
            wall_height = 1.0 # in meters
            extra_width = 0.6 # in meters
            world = client.generate_opendrive_world(
                data, carla.OpendriveGenerationParameters(
                    vertex_distance=vertex_distance,
                    max_road_length=max_road_length,
                    wall_height=wall_height,
                    additional_width=extra_width,
                    smooth_junctions=True,
                    enable_mesh_visibility=True))
    else:
```

```
print('file not found.')
elif args.osm_path is not None:
    if os.path.exists(args.osm_path):
        with open(args.osm_path, encoding='utf-8') as
            od_file:
                try:
                    data = od_file.read()
                except OSError:
                    print('file could not be readed.')
                    sys.exit()
                print('Converting OSM data to opendrive')
                xodr_data = carla.Osm2Odr.convert(data)
                print('load opendrive map.')
                vertex_distance = 2.0 # in meters
                max_road_length = 500.0 # in meters
                wall_height = 0.0 # in meters
                extra_width = 0.6 # in meters
                world = client.generate_opendrive_world(
                    xodr_data, carla.OpendriveGenerationParameters
                    (
                        vertex_distance=vertex_distance,
                        max_road_length=max_road_length,
                        wall_height=wall_height,
                        additional_width=extra_width,
                        smooth_junctions=True,
                        enable_mesh_visibility=True))
                else:
                    print('file not found.')

else:
    world = client.get_world()

settings = world.get_settings()

if args.no_rendering:
    print('disable rendering.')
    settings.no_rendering_mode = True
```

```
        elif args.rendering:
            print('enable rendering.')
            settings.no_rendering_mode = False

        if args.no_sync:
            print('disable synchronous mode.')
            settings.synchronous_mode = False

        if args.delta_seconds is not None:
            settings.fixed_delta_seconds = args.
                delta_seconds
        elif args.fps is not None:
            settings.fixed_delta_seconds = (1.0 / args.fps
                ) if args.fps > 0.0 else 0.0

        if args.delta_seconds is not None or args.fps
            is not None:
            if settings.fixed_delta_seconds > 0.0:
                print('set fixed frame rate %.2f milliseconds
                    (%d FPS)' %
                    (1000.0 * settings.fixed_delta_seconds,
                     1.0 / settings.fixed_delta_seconds))
            else:
                print('set variable frame rate.')
                settings.fixed_delta_seconds = None

        if args.tile_stream_distance is not None:
            settings.tile_stream_distance = args.
                tile_stream_distance
        if args.actor_active_distance is not None:
            settings.actor_active_distance = args.
                actor_active_distance

    world.apply_settings(settings)

    if args.weather is not None:
        if not hasattr(carla.WeatherParameters, args.
```

```
        weather):
    print('ERROR: weather preset %or not found.' %
          args.weather)
else:
    print('set weather preset %r.' % args.weather)
world.set_weather(getattr(carla.
                      WeatherParameters, args.weather))

if args.inspect:
    inspect(args, client)
if args.list:
    list_options(client)
if args.list_blueprints:
    list_blueprints(world, args.list_blueprints)

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('\nCancelled by user. Bye!')
    except RuntimeError as e:
        print(e)
```