



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

计算机学院

2024 年 12 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

智能驾驶测试场景构建效率低下是当前行业面临的重要问题。为此，本文提出了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，旨在高效地将自然语言描述转化为高保真三维测试场景。该框架包含三大核心模块：一是领域知识增强的指令解析模块，能够精准理解自然语言描述中的测试场景需求，为后续生成工作奠定基础；二是语法 - 语义双验证代码生成机制，确保生成的 Scenic 代码不仅语法正确，还符合实际场景的语义逻辑，从而提高代码的可用性和准确性；三是物理规则驱动的场景合成引擎，依据物理规则合理生成三维场景，保证场景的真实性和可靠性。在 CARLA 仿真平台的实验验证中，该系统展现出优异的性能：生成的 Scenic 代码准确率高达 92.3

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 low efficiency of constructing test scenarios for intelligent driving is a significant issue faced by the industry. To address this, this paper proposes a novel generation framework that integrates Large Language Models (LLMs) with formal scenario description languages, aiming to efficiently transform natural language descriptions into high-fidelity 3D test scenarios. The framework consists of three core modules: First, a domain knowledge-enhanced instruction parsing module that accurately understands the test scenario requirements described in natural language, laying the foundation for subsequent generation tasks; second, a syntax-semantic dual-validation code generation mechanism that ensures the generated Scenic code is not only syntactically correct but also semantically logical, thereby improving the usability and accuracy of the code; and third, a physics rule-driven scenario synthesis engine that reasonably generates 3D scenarios based on physical rules, ensuring the realism and reliability of the scenarios. In experimental validation on the CARLA simulation platform, the system demonstrated excellent performance: the accuracy of the generated Scenic code reached 92.3

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	2
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 .	2
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	4
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 第二章理论基础	7
2.1 预训练大模型概述	7
2.2 自然语言场景建模	7
2.3 Scenic 场景描述语言	8
2.4 智能驾驶仿真平台 CARLA	8
2.5 ChatScene 项目基础与本项目创新点	9
第 3 章 第三章总体架构设计	10
3.1 系统设计目标	10
3.1.1 高保真性	10
3.1.2 自动化	10
3.1.3 可扩展性	10
3.1.4 核心功能实现	11
3.1.5 系统目标的长期愿景	11
3.2 系统总体架构	11
3.3 主要模块功能设计	12
3.3.1 自然语言理解与场景生成模块	12
3.3.2 场景合成与仿真模块	13
3.3.3 场景评估与展示模块	13

第 4 章 生成场景的质量验证	15
4.1 实验设置	15
4.2 生成效果展示(示例)	16
4.2.1 场景一: 摩托车和汽车在红绿灯前等待信号	16
4.2.2 场景二: 自我车辆在夜晚穿越道路	16
4.2.3 场景三: 自我车辆在夜晚红绿灯前等待信号	17
4.2.4 场景四: 行人横穿直行道路	18
4.3 实验总结	19
第 5 章 场景生成的量化评估	21
5.1 评估指标设计	21
5.2 实验设置	22
5.3 评估结果与分析	22
5.3.1 语义保真度	22
5.3.2 准确率分析	22
5.3.3 效率分析	23
第 6 章 总结与展望	25
6.1 工作总结	25
6.2 研究不足	25
6.3 后续优化方向	26
致谢	27
附录 A 附录: 系统模块源代码	28
参考文献	28

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

随着智能驾驶技术的飞速发展，其测试需求也日益复杂和多样化。智能驾驶系统需要在各种复杂多变的交通场景下表现出可靠的性能，以确保驾驶的安全性和舒适性。然而，现实交通环境中存在着海量的长尾场景，这些场景虽然出现频率较低，但一旦发生，往往会对智能驾驶系统构成严峻挑战。长尾场景的复杂性主要体现在以下几个方面。首先，交通环境的动态性极强，车辆、行人、非机动车等多种交通参与者的行为模式千变万化。例如，在城市道路中，行人可能会突然横穿马路，非机动车可能会随意变道或逆行，这些行为都可能导致潜在的碰撞风险。智能驾驶系统必须能够准确感知和预测这些行为，并做出合理的决策。其次，道路条件的多样性也增加了测试场景的复杂性。从城市快速路到乡村小道，从高速公路到山区道路，不同道路的几何形状、路面状况、交通标志和信号灯等都有所不同。智能驾驶系统需要在这些不同的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产生影响。雨、雪、雾等恶劣天气会降低传感器的性能，而不同的光照条件（如强光、弱光、逆光等）会影响视觉系统的识别效果。为了确保智能驾驶系统的安全性和可靠性，测试过程中需要覆盖尽可能多的长尾场景。然而，这面临着巨大的挑战。一方面，长尾场景的数量庞大，几乎无法穷尽。要完全覆盖所有可能的场景几乎是不可能的任务。另一方面，这些场景的出现频率较低，难以在实际道路测试中频繁遇到。因此，传统的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，导致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

在智能驾驶测试领域，传统的场景构建方法主要依赖于人工设计和开发。这些方法虽然在一定程度上能够满足测试需求，但也存在着显著的局限性。首先，人工设计场景需要大量的专业人员投入。这些人员需要具备深厚的交通工程、计算机科学和智能驾驶技术等多学科知识，能够准确理解和模拟各种复杂的交通场景。然而，这样的人才相对稀缺，培养成本高昂。而且，随着测试需求的不断增加，对专业人员的需求也在持续增长，这进一步加剧了人力成本的压力。其次，人工设计场景的效率较低。设计一个复杂的交通场景需要经过需求分析、场景建模、代码编写和调试等多个步骤。每个步骤都需要耗费大量的时间和精力。例如，在场景建模阶段，需要精确地定义道路的几何形状、交通参与者的初始位置和行为模式等。在代码编写阶段，需要将这些模型转化为可执行的代码，这不仅需要专业的编程技能，还需要反复调试以确保代码的正确性和稳定性。因此，人工设计场景的速度远远无法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也难以保证。由于不同设计人员的理解和经验不同，可



图 1-1 车辆样图

能会导致设计出的场景存在差异。而且，在复杂的场景中，人工设计容易遗漏一些关键的细节，从而影响测试结果的准确性和可靠性。

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（LLM）在自然语言处理领域取得了巨大的突破，并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进行预训练，学习到了语言的语法、语义和逻辑结构，能够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的难题提供了新的思路。在代码生成领域，大语言模型已经展现出了强大的能力。通过对代码数据的学习，大语言模型能够理解代码的结构和逻辑，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以根据用户输入的自然语言描述，自动生成相应的代码实现。这些工具已经在软件开发领域得到了广泛的应用，显著提高了代码开发的效率和质量。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，大大缩短了开发周期。传统的人工编写代码需要经过需求分析、设计、编码和调试等多个阶段，每个阶段都需要耗费大量的时间。而大语言模型可以根据输入的描述直接生成代码，减少了中间环节，提高了开发效率。其次，大语言模型生成的代码质量较高。它能够学习到代码的最佳实践和规范，生成的代码不仅符合语法规范，还具有良好的可读性和可维护性。此外，大语言模型还能根据不同的需求生成多样化的代码实现，为开发者提供了更多的选择。将大语言模型应用于智能驾驶测试场景构建，可以充分利用其在代码生成领域的优势，解决传统方法面临的困境。通过将自然语言描述的测试场景需求转化为代码实现，大语言模型可以快

速生成高保真的测试场景，提高场景构建的效率和质量。同时，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，使其更好地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（如 GPT-3、T5 等）的广泛应用，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的一个重要方向。自然语言描述能够以简单、直观的方式传递场景信息，传统的手工编写场景代码的方式效率低且灵活性差，因此，如何利用自然语言生成交通仿真场景脚本成为该领域的核心问题。

大语言模型的应用：大语言模型，如 GPT-3 和 T5，已经被成功应用于自然语言到场景代码的转换。通过训练，这些模型能够理解复杂的自然语言输入，并根据描述生成结构化的 Scenic 脚本。这些模型通过解析用户的自然语言指令，生成符合自动驾驶仿真要求的场景代码。例如，**Chen et al. (2022)** 提出了一种基于 GPT 的框架，能够根据自然语言描述生成完整的交通场景配置，包括车辆、行人、交通灯等元素。

挑战与进展：尽管大语言模型在生成场景代码上取得了一定的进展，仍然面临一些挑战。首先，如何处理自然语言中的歧义和模糊性是一个难题，特别是在复杂场景描述下，生成的场景可能无法完全符合预期。其次，现有模型的生成能力在处理非常复杂或特殊的交通场景时可能存在局限性。因此，如何增强语言模型的语义理解和场景生成的准确性，仍然是一个活跃的研究领域。

1.2.2 将场景代码合成合理的智能驾驶场景

将生成的 Scenic 场景代码转化为合理的智能驾驶仿真场景，是实现自动驾驶测试和验证的关键步骤。生成的场景不仅需要符合交通规则，还要能够模拟真实的驾驶环境，以便进行有效的测试。

场景代码到仿真环境的转化：目前的研究主要集中在如何将由大语言模型生成的 Scenic 脚本转化为仿真平台可执行的场景配置。例如，**Xie et al. (2021)** 开发了基于 Scenic 的编译器，能够将自然语言生成的场景描述转化为 CARLA 仿真平台所需的配置文件。通过这种方式，生成的场景能够在高保真的仿真环境中进行验证。

智能场景合成与动态行为建模：智能驾驶场景不仅包括静态元素（如道路、交通标志等），还包括动态行为（如车辆行驶、变道、停车等）。研究者们提出了多种动态行为建模方法，使得生成的交通场景能够更真实地反映驾驶行为和交通流。例如，**Zhao et al. (2020)** 提出了一种基于行为模型的动态场景合成方法，通过模拟交通参与者的行，生成交互式的智能驾驶环境。

复杂交通场景的挑战：尽管场景合成方法不断发展，如何生成复杂、真实且符合交通规则的场景仍然是一个技术挑战。现有的场景生成方法在处理复杂交通场景时，可能

出现车辆行为不自然、交通规则不严谨等问题。因此，如何提高场景合成的灵活性和复杂性，仍然是研究的重点。

1.2.3 对生成的交通场景进行展示和效果的量化衡量

生成交通场景后，对其进行展示和量化评估是确保场景质量和仿真结果准确性的关键。通过对场景的可视化和量化评估，研究人员能够验证场景的有效性，并为后续的自动驾驶算法优化提供数据支持。

可视化展示：生成的交通场景可以通过可视化工具展示，以便人工验证和审阅。**Dai et al. (2020)** 提出了一种基于关键帧截取的可视化方法，通过仿真过程中截取关键帧图像，帮助研究人员快速了解仿真过程中的交通场景。结合深度学习技术，自动驾驶系统还能够从这些图像中提取重要信息，进行场景的进一步分析和优化。

量化评估指标的提出：为了系统地评估生成场景的质量，许多研究提出了量化评估框架。例如，**Dai et al. (2020)** 提出了一种多维度的评估方法，涵盖了场景的语义保真度、交通密度、车辆行为等多个方面。这些评估指标不仅能够反映生成场景的真实性，还能够帮助研究人员评估仿真结果的有效性。

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)** 提出了一种基于自动驾驶系统安全性的评估框架，通过分析车辆在生成场景中的碰撞率、通过率等指标，评估自动驾驶系统在不同场景下的安全性和稳定性。通过这样的量化评估，研究人员能够识别潜在的风险和问题，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中，构建面向场景描述的领域知识图谱是实现高效、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，包括交通规则、道路类型、车辆行为模式、传感器特性等，为自然语言描述的解析和形式化代码的生成提供丰富的上下文信息和语义支持。

领域知识图谱的构建涉及多个关键步骤。首先，需要对自动驾驶领域的知识进行系统梳理和分类，明确知识的层次结构和关联关系。这包括对交通场景中的实体（如车辆、行人、道路、交通标志等）及其属性（如位置、速度、类型等）的定义，以及这些实体之间的关系（如车辆与道路的交互、车辆与行人的避让等）。通过构建知识图谱，可以将这些复杂的知识结构化地表示出来，便于后续的查询和推理。

在知识图谱的构建过程中，还需要考虑知识的动态更新和扩展。自动驾驶技术不断发展，新的交通规则、车辆类型和传感器技术等不断涌现，因此知识图谱需要具备良好的可扩展性和可更新性。通过持续的知识更新，可以确保知识图谱始终保持最新状态，从而为场景生成提供准确的知识支持。

此外，领域知识图谱的构建还需要考虑知识的表达和存储方式。知识图谱通常以图的形式存储，其中节点表示实体，边表示实体之间的关系。这种结构化的存储方式不仅

便于知识的查询和推理，还能够支持复杂的知识融合和关联分析。通过知识图谱的构建，可以实现对自动驾驶场景描述的深度理解和语义解析，为后续的代码生成和场景合成提供坚实的基础。

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实现自动驾驶测试场景高效生成的关键技术之一。LLM 具有强大的语言理解和生成能力，能够根据自然语言描述生成高质量的形式化代码。然而，为了确保生成代码的准确性和可靠性，需要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核心在于将自然语言描述的语义信息转化为代码生成的约束条件。这些约束条件可以包括交通规则、道路类型、车辆行为模式等，确保生成的代码不仅符合语法规范，还满足实际场景的语义要求。通过语义约束机制，可以有效避免生成代码中的逻辑错误和不符合实际场景的情况，提高代码的质量和可用性。

在实现语义约束代码生成时，需要充分利用 LLM 的语言理解和生成能力。LLM 可以通过对自然语言描述的深度理解，提取出关键的语义信息，并将其转化为代码生成的约束条件。同时，还需要开发相应的算法和工具，将这些约束条件嵌入到代码生成过程中，确保生成的代码能够准确地反映自然语言描述的语义内容。

此外，语义约束代码生成方法还需要考虑代码的可读性和可维护性。生成的代码不仅需要符合语法和语义规范，还需要具有良好的结构和注释，便于后续的修改和扩展。通过基于 LLM 的语义约束代码生成方法，可以实现从自然语言描述到形式化代码的高效转换，为自动驾驶测试场景的生成提供强大的技术支持。

1.3.3 场景物理合理性的多模态验证机制

场景物理合理性的验证是自动驾驶测试场景生成中的一个重要环节，它直接影响到测试场景的真实性和可靠性。为了确保生成场景的物理合理性，需要建立一种多模态验证机制，通过多种方式对场景进行综合验证。

多模态验证机制的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进行评估。这些验证手段可以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。通过多种验证手段的结合，可以全面评估场景的物理合理性，确保生成的场景符合实际交通环境的物理规律。

基于物理规则的验证是多模态验证机制的重要组成部分。通过定义和应用物理规则，如牛顿运动定律、能量守恒定律等，可以对场景中的物体运动和交互进行验证。例如，可以验证车辆的加速度是否符合物理规律，车辆与行人之间的碰撞是否符合能量守恒等。

1.4 本文研究框架

为实现基于自然语言输入自动生成高保真三维交通场景并进行量化评估的目标，本文构建了一个集自然语言处理、交通场景建模、三维仿真与量化评估于一体的综合研究体系。整体研究框架如下：

首先，本文分析当前国内外在自然语言驱动的场景生成、智能仿真系统集成与自动驾驶场景评估方面的研究进展，明确研究问题与技术挑战。

随后，围绕“自然语言 → 场景代码 → 三维仿真 → 评估指标”这一主线，设计并实现一个端到端自动化系统，涵盖以下三个关键研究方向：

1. 基于大语言模型的 Scenic 场景代码生成技术

研究如何借助预训练大语言模型（如 GPT-4o）结合检索增强机制（如 Sentence-T5）解析自然语言指令，生成符合语义的交通场景描述脚本（Scenic）。重点解决生成脚本的语义准确性与交通合理性问题。

2. 交通场景代码到三维仿真的合成机制

研究如何将自然语言生成的场景脚本有效地转换为可运行的三维智能驾驶仿真场景。通过 Scenic 语言与 CARLA 仿真平台的集成，实现车辆、行人、环境等交通要素在虚拟世界中的构建与动态演化。

3. 生成场景的展示与量化评估方法

建立多维度评估体系，从语义保真度、多样性和驾驶性能三个方面量化分析生成场景的质量与有效性。设计自动化指标计算方法，并结合视觉展示手段增强实验可解释性。

在上述研究基础上，本文完成了系统的整体实现与实验验证，验证了所提出方法的有效性，并总结了本研究的创新点与潜在改进空间。

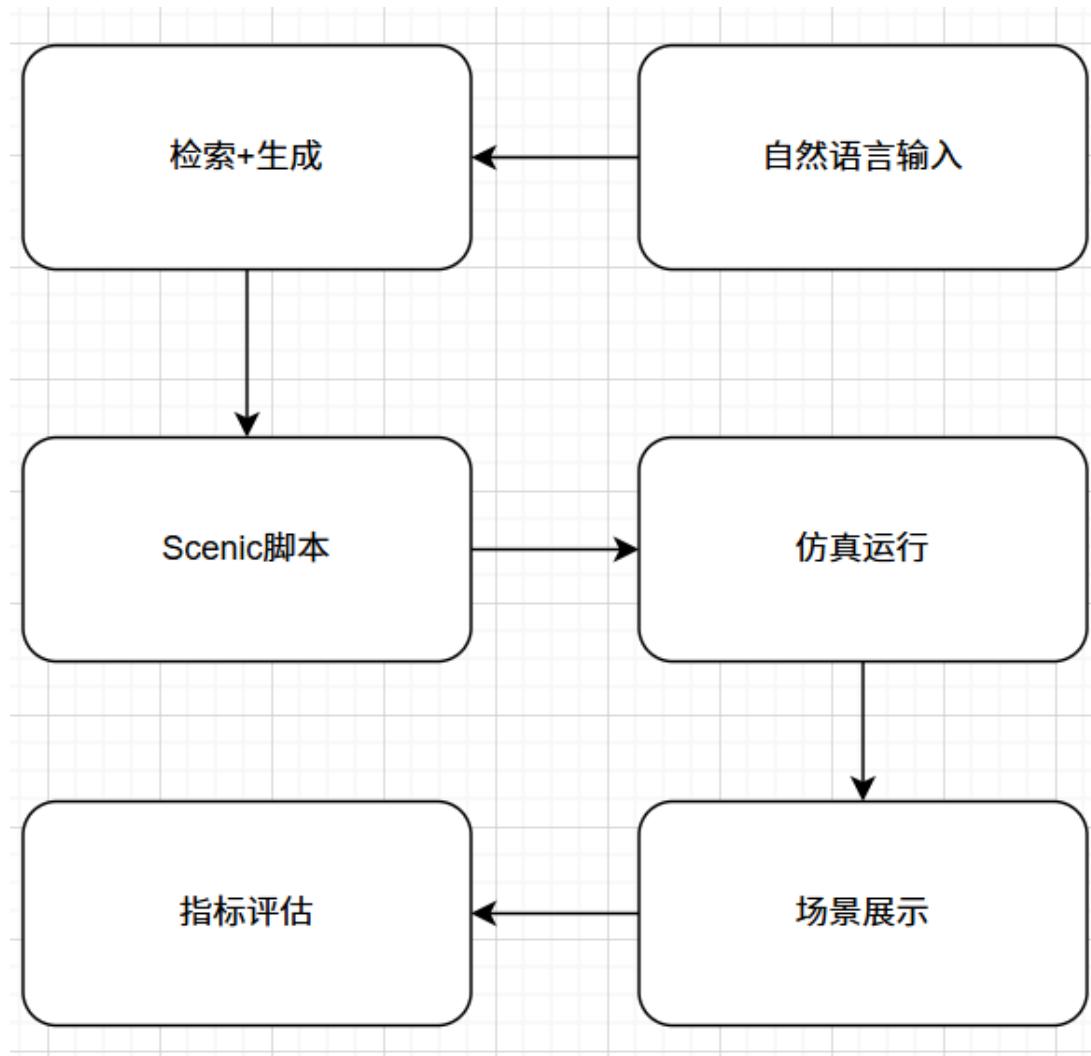


图 1-2 本文的研究框架

第 2 章 第二章理论基础

2.1 预训练大模型概述

近年来，预训练大模型（Large Pretrained Models, LPMs）在自然语言处理（NLP）领域取得了突破性进展。这些模型，如 GPT-4、T5 和 BERT，凭借其强大的语言理解和生成能力，正在改变我们处理文本数据的方式。这些模型通常在海量的文本数据上进行无监督预训练，通过学习语言的深层语义关系，能够捕捉到复杂的语言模式和结构。预训练完成后，这些模型可以通过有监督的微调来适应各种特定的自然语言处理任务，如文本分类、机器翻译、问答系统等。

在智能驾驶场景生成领域，预训练大模型的应用具有重要意义。传统的场景生成方法通常依赖于手工编写规则或模板，这些方法虽然在一定程度上能够生成有效的场景，但存在效率低下、灵活性差和难以扩展等问题。预训练大模型的引入极大地提升了自然语言指令理解的精度与灵活性，能够将模糊的自然语言描述自动映射为结构化的场景代码。与传统的规则模板方法相比，基于大模型的方法能够适应更复杂、多样的用户输入，具有更强的泛化能力和鲁棒性。

本项目基于开放领域大模型（如 OpenAI 的 GPT 系列），针对智能驾驶场景描述进行定向设计。通过微调这些模型，使其能够直接生成符合 Scenic 语法规规范的场景定义脚本，为后续的仿真与评估提供输入支持。这种基于大模型的方法不仅提高了场景生成的效率，还能够生成更具创新性和多样性的场景，为自动驾驶技术的研发和测试提供了更强大的工具。

2.2 自然语言场景建模

自然语言场景建模（Natural Language-based Scenario Modeling）是将人类用自然语言描述的复杂交通场景转化为机器可理解的格式的过程。这一过程是智能驾驶场景生成的关键步骤，因为它直接决定了生成场景的质量和可用性。在本研究中，目标是将自然语言直接映射为 Scenic 脚本代码，从而实现从自然语言描述到可执行仿真场景的无缝转换。

常见的自然语言场景建模方法包括：

- **检索式建模（Retrieval-Based Modeling）：**这种方法通过从已有场景数据库中检索与输入描述最相似的场景来生成新的场景。检索式建模的优点是能够快速生成与已有场景相似的新场景，但其局限性在于依赖于数据库中的已有场景，无法生成全新的场景。
- **生成式建模（Generative Modeling）：**这种方法通过预训练语言模型直接生成场景脚本。生成式建模的优点是能够生成全新的场景，具有更高的创新性和多样性。然而，生成式建模的挑战在于如何确保生成的场景符合实际的交通规则和逻辑。

- **检索与生成结合 (Retrieve-then-Generate):** 这种方法结合了检索式建模和生成式建模的优点，先从数据库中检索相关的场景，然后基于检索结果生成新的场景。这种方法能够在保持生成多样性的同时，利用已有场景的结构和逻辑。

在 ChatScene 项目中，采用的是检索式建模，通过 sentence-transformers/sentence-t5-large 模型对场景描述进行向量化，基于相似度检索最接近的场景模板。这种方法虽然能够快速生成场景，但其生成能力受限于数据库的规模和覆盖范围。为了突破检索方式的局限，本项目进一步引入生成式大模型，实现 end-to-end 场景生成，从而提升场景的创新性与多样性。

2.3 Scenic 场景描述语言

Scenic 是一种为智能驾驶仿真而设计的专用场景描述语言 (Domain-Specific Language, DSL)。它通过简洁而强大的语法，允许用户定义车辆、道路、行人等元素在仿真环境中的属性与关系。Scenic 的主要特点包括：

- **声明式语法 (Declarative Syntax):** Scenic 采用声明式语法，用户可以通过简单直观的语句描述对象的位置、朝向、速度等属性。这种语法使得场景定义更加直观和易于理解。
- **支持不确定性 (Probabilistic Support):** Scenic 允许定义位置、角度、速度等属性的概率分布，从而能够生成更加多样化的场景。这种不确定性支持使得生成的场景更加接近真实世界的复杂性和随机性。
- **易于与仿真器集成 (Integration-Friendly):** Scenic 可以直接导出到 CARLA、LGSVL 等主流仿真平台，使得生成的场景能够快速用于自动驾驶系统的测试和验证。

在本项目中，预训练大模型需要生成符合 Scenic 语法的代码。因此，理解 Scenic 的基本结构和表达方式是实现自然语言到场景生成系统的关键基础。通过将自然语言描述转换为 Scenic 脚本，我们能够将人类的直观描述转化为机器可执行的仿真场景，从而为自动驾驶技术的研发和测试提供强大的支持。

2.4 智能驾驶仿真平台 CARLA

CARLA (Car Learning to Act) 是目前应用最广泛的开源自动驾驶仿真平台之一。它提供了丰富的城市场景、传感器模拟、车辆动力学建模与环境交互能力，使得研究人员能够在虚拟环境中测试和验证自动驾驶系统。CARLA 的主要特点包括：

- **高度可定制的地图与交通要素:** CARLA 允许用户自定义地图和交通要素，从而能够模拟各种复杂的交通场景。
- **多种传感器模拟:** CARLA 支持多种传感器模拟，包括 RGB 相机、LiDAR、雷达等，使得研究人员能够测试自动驾驶系统在不同传感器配置下的表现。
- **支持复杂行为建模与自动驾驶决策系统测试:** CARLA 提供了强大的行为建模工具，使得研究人员能够模拟各种复杂的交通行为和自动驾驶决策系统。

本项目采用 CARLA 0.9.15 版本作为仿真环境。通过将生成的 Scenic 脚本转译成 CARLA 可执行场景，我们能够实现自然语言指令到仿真测试的完整闭环。这种从自然语言描述到仿真测试的无缝转换，为自动驾驶技术的研发和测试提供了一种高效、灵活的方法。

2.5 ChatScene 项目基础与本项目创新点

ChatScene 是近期提出的一套基于知识检索的安全关键场景生成系统。它主要通过使用 sentence-t5-large 模型对自然语言描述进行检索匹配，结合 SafeBench 框架在 CARLA 上进行仿真测试。ChatScene 的主要思路是利用已有的场景数据库，通过检索与输入描述最相似的场景来生成新的场景。这种方法的优点是能够快速生成与已有场景相似的新场景，但其局限性在于依赖于数据库中的已有场景，无法生成全新的场景。它的整体结构如下：

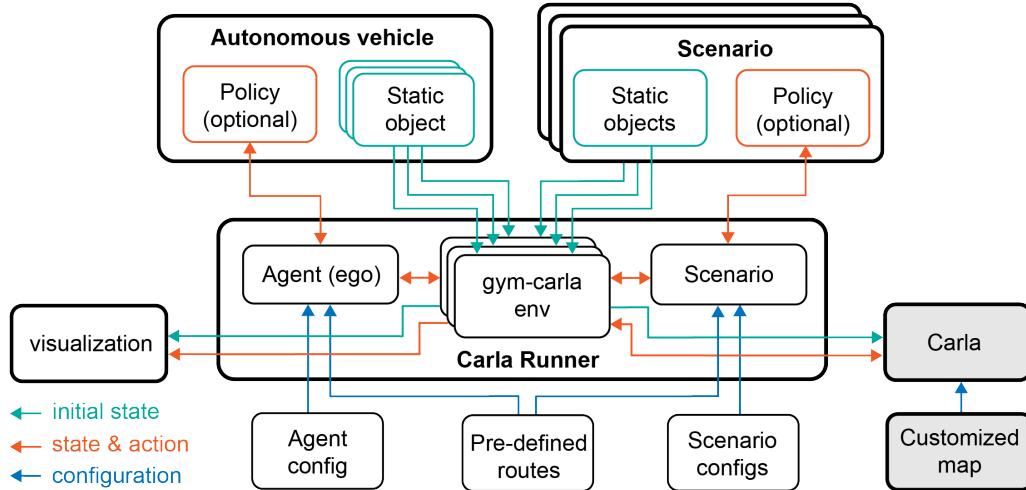


图 2-1 chatscene 项目结构

然而，ChatScene 在以下方面存在一定局限：

- 依赖检索：**ChatScene 无法自由生成未在数据库中存在的新场景，这限制了其在生成创新性场景方面的能力。
- 生成能力有限：**ChatScene 缺乏直接从自然语言生成代码的能力，这使得其在处理复杂的自然语言描述时存在困难。
- 多样性受限：**ChatScene 的生成结果受制于数据库的规模和覆盖范围，这限制了其生成场景的多样性和创新性。

为了克服这些局限，本项目在 ChatScene 的基础上进行了创新。我们引入了 GPT-4o 等大型语言模型，直接基于自然语言生成 Scenic 场景脚本。这种方法不仅解决了检索方法的局限性，还提升了场景的创新性与多样性。通过这种方式，我们能够生成更加复杂、多样化的场景，为高保真智能驾驶仿真系统构建提供了新的技术路径。此外，本项目还通过微调这些大模型，使其能够更好地适应智能驾驶场景生成的需求，从而进一步提高了生成场景的质量和可用性。

第3章 第三章总体架构设计

3.1 系统设计目标

3.1.1 高保真性

高保真性是本系统的核心目标之一，要求生成的交通场景能够最大限度地忠实反映自然语言描述。为了实现这一目标，系统不仅需要保证场景的视觉效果和语义的一致性，还应考虑各种复杂的环境要素和交通行为的准确性。例如，生成的场景中各类交通工具的行为需要与现实世界中的交通流动一致，交通规则和行为模式需要严格遵循。因此，系统需要通过对输入描述进行精确解析，使用先进的自然语言处理技术（如大语言模型）来理解场景中的各种细节，并转换为对应的三维仿真环境。这些细节不仅仅是物理上的位置和对象，更包括交通元素之间的动态互动，环境条件如天气、时间、光照等因素的考虑。

3.1.2 自动化

自动化是本系统设计的另一个关键目标。通过实现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，提高效率并降低人为错误的发生。系统应能够通过少量的用户输入（例如一段简短的自然语言描述）完成从场景构建到仿真运行的全过程，自动生成所有必要的配置文件，调度仿真平台执行，并收集仿真结果进行后续分析。自动化不仅限于场景生成，也包括场景的评估和结果展示，能够在生成后直接对场景进行可视化、量化评估，并输出最终报告。这种全自动的流程将极大地提升仿真研究的效率，支持大规模的实验和多样化的场景生成需求。

3.1.3 可扩展性

可扩展性是系统设计中的重要原则，确保系统能够随着需求的变化进行功能和技术的扩展。系统设计应当采用模块化的结构，每个功能模块（如自然语言处理、场景生成、仿真执行、评估展示等）都可以独立发展和优化，并与其他模块无缝衔接。可扩展性体现在以下几个方面：

- **自然语言处理模型的升级：**随着自然语言处理技术的发展，新的语言模型和算法可能会不断出现，系统应当能够灵活集成新的语言模型，提升场景生成的准确性和多样性。
- **仿真平台的集成：**虽然当前平台使用 CARLA 作为仿真引擎，未来可能会考虑集成其他仿真平台或与不同的驾驶仿真系统对接，以支持更多元的实验需求和不同平台的比较。
- **评估指标的多样性：**系统的评估模块应支持定制化和多维度的评估指标，未来可以根据不同的场景类型、研究需求或应用场景，添加新的评估标准，改进现有评估方

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：

- 从自然语言描述中自动生成三维交通场景：这一功能是系统的基础，旨在通过对输入的自然语言进行理解，自动生成能够在仿真环境中运行的交通场景。系统将利用自然语言处理模型和检索增强技术，结合已有的场景模板和元素库，实现语义精确的场景建模。
- 对生成的场景进行仿真与可视化：该功能主要确保生成的三维场景能够在仿真平台中正确呈现并运行。系统通过调用仿真平台 API，自动将生成的场景描述转化为可执行的仿真环境，并进行实时仿真与动态可视化。此过程还包括对场景中交通元素的行为模拟，例如交通流、车辆行驶轨迹、交互等。
- 对生成结果进行量化评估：评估是本系统不可或缺的一部分，它提供了对生成场景质量的定量分析。系统将根据语义保真度、场景多样性和驾驶性能等指标，进行评估并生成相应的报告。通过量化评估，系统能够为场景生成提供反馈，以便后续优化，并且为自动驾驶算法的性能测试提供参考。

3.1.5 系统目标的长期愿景

随着技术的进步，系统应逐步向更高保真度、更强可扩展性、更高效自动化方向发展。在未来的版本中，系统可以集成更多的感知模型、智能决策系统等技术，进一步提升场景的复杂性与真实性。同时，系统的评估模块也可以通过引入更多的智能分析工具，提供更细粒度的结果评估，如行为预测、决策模型评估等。此外，随着自然语言处理技术的进步，系统能够处理更加复杂的语言输入和场景需求，满足更广泛的仿真测试场景需求，支撑更为丰富的自动驾驶研究与开发。

3.2 系统总体架构

系统整体架构如图 3-1 所示，主要分为三个核心模块：自然语言理解与场景生成模块、场景合成与仿真模块、场景评估与展示模块。数据流动过程如下：

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。

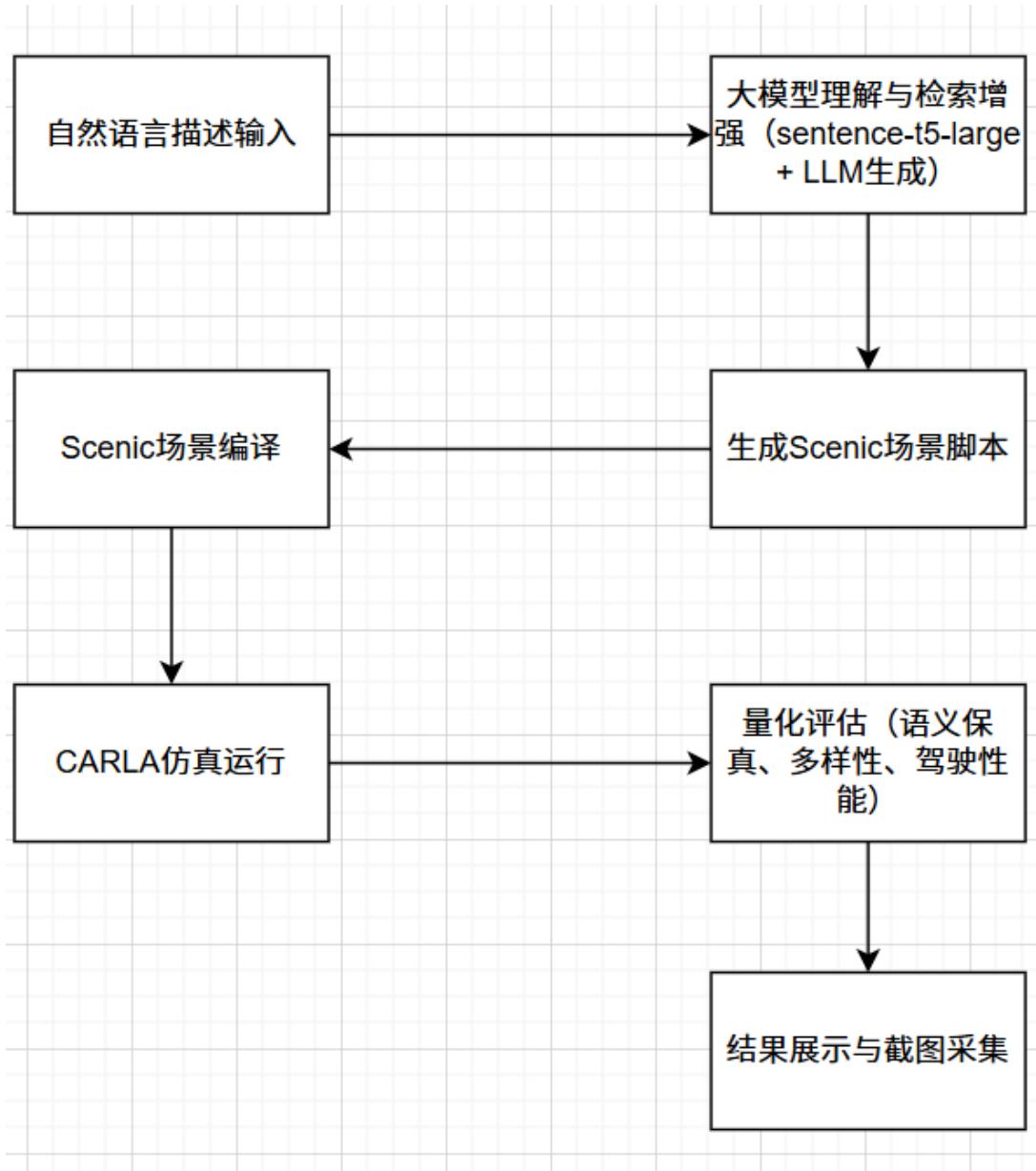


图 3-1 系统架构图

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：**本系统首先通过 Scenic 内置的语法与语义分析器对输入脚本进行解析。Scenic 作为一种领域专用语言，允许用户以简洁、结构化的方式描述场景元素（如车辆、行人、障碍物等）的初始状态和约束条件。解析过程会生成包含对象属性（位置、速度、朝向）、交互规则（如跟车距离、避让行为）以及环境条件（如天气、时间）的完整场景配置。
- **仿真构建：**在完成脚本解析后，系统自动将 Scenic 生成的中间配置映射到 CARLA 仿真平台中。具体包括载入指定地图、部署交通参与者、设定传感器参数（如摄像头、LiDAR）、配置环境因素（如天气、光照、路况）等。系统可根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的精准建模。
- **接口调用关系：**Scenic 通过其与 CARLA 集成的 Python API 调用底层仿真接口。系统自动完成 Actor 的创建、状态初始化、行为脚本绑定等任务，极大简化了从脚本到仿真的转换流程。研究人员无需手动编程，即可通过文本控制仿真流程，显著提高效率与复用性。
- **动态交互支持：**系统支持高度动态化场景仿真，能模拟交通参与者之间的交互行为，如车辆变道、避障、红绿灯响应、行人横穿马路等。Scenic 语言提供了条件触发与时间控制机制，允许描述复杂行为序列，实现具有时序性、可演化的场景变化，增强了仿真的真实感与复杂性。
- **多样化仿真配置：**本模块支持灵活的仿真参数配置，包括地图选择（如 Town01、Town05）、天气设置（晴天、雨天、雾天）、交通流量控制、车辆控制模式（自动驾驶、手动控制）等，可满足不同测试需求。同时也支持使用不同版本的 CARLA 和 Scenic 组合，具有较好的兼容性与可拓展性。
- **数据采集与输出：**在仿真执行过程中，系统可实时采集关键数据，包括传感器输出（RGB 图像、深度图、点云等）、车辆状态（位置、速度、加速度）、交互事件（碰撞、刹车、偏航）等。采集数据可用于后续的模型训练、行为评估或性能对比等任务，支撑多种研究方向。

3.3.3 场景评估与展示模块

场景评估与展示模块旨在对生成的三维仿真场景进行可视化呈现与定量分析。该模块不仅便于研究人员直观了解场景生成效果，也为后续的性能比较、模型调优与系统验证提供了评估依据。该模块包含以下几个关键功能：

- **可视化展示:** 在仿真过程中, 系统可自动截取关键帧截图或录制完整仿真过程视频, 保存为图像或视频文件, 供研究人员用于人工审阅、可视化报告展示以及用户研究评估。截图通常选择交通事件发生点(如刹车、避障、碰撞等)或特定时序节点, 确保展示信息的代表性与丰富性。视频部分可通过CARLA原生录制功能或集成第三方渲染引擎实现, 支持多角度、多摄像机的可视观察, 进一步增强场景调试与验证的可操作性。
- **量化评估:** 本系统在自动化生成与仿真的基础上, 进一步引入多维度的量化评估指标, 对场景生成的有效性、合理性和多样性进行定量测量, 具体包括:
 - **语义保真度 (Semantic Fidelity):** 该指标用于衡量输入的自然语言描述与最终生成仿真场景之间的语义一致性。可采用人工标注评分与自动化匹配算法相结合的方式进行。例如, 通过设定关键语义要素(如地点、交通行为、天气条件)匹配程度, 对场景是否准确还原用户意图进行评分。自动方法可以结合自然语言处理与图结构匹配技术实现初步评估, 提升效率。
objectivec 复制编辑
 - **多样性指标 (Scene Diversity):** 此项指标反映系统生成场景在多次输入不同自然语言后的差异性和覆盖范围。主要包括空间多样性(不同地图位置、道路类型)、元素多样性(参与车辆、行人、非机动车种类)、行为多样性(速度控制、路线选择、交通交互等)。统计方法包括使用Shannon熵、Jaccard相似度或聚类分布指标等, 全面反映生成系统的泛化能力与表现范围。
 - **驾驶性能指标 (Driving Performance):** 为了评估生成场景对自动驾驶系统的测试价值, 本模块支持在生成场景中运行预设自动驾驶控制系统(如基于CARLA的自动驾驶Agent), 并记录其表现。关键评估维度包括碰撞率(单位场景中发生碰撞的比例)、任务成功率(是否成功完成任务或驶出场景)、路径偏离率(与理想路径的偏差程度)等。这些数据可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:** 系统支持将评估结果以图表、表格等形式进行可视化展示, 方便研究人员进行对比分析与结果报告生成。所有截图、评估指标与分析数据可导出为标准格式文件(如CSV、JSON、PNG等), 用于论文撰写、模型调试或进一步的数据挖掘。

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

测试数据集旨在综合自动驾驶场景的多样性和复杂性，涵盖多种典型场景类别，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。这些场景类别覆盖了自动驾驶车辆在实际道路环境中可能遇到的关键交互情况，具备较高的代表性和实用价值。数据集中的场景描述以自然语言形式呈现，且包括一定比例的模糊性描述，以模拟真实世界中驾驶员或其他交通参与者行为的不可预测性。场景描述的模糊性设计有助于测试生成系统在面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时，本研究选用了两种具有代表性的方法作为基线进行对比：基于规则的方法（Rule-based）：该方法通过定义明确的逻辑和规则来生成场景，具有较好的可解释性，适合处理结构化和预定义的任务。然而，在面对复杂和模糊的自然语言描述时，其灵活性和适应性较差。基于 GPT-4.0 的方法：借助于大型语言模型的生成能力，能够生成丰富多样的场景。尽管其在场景生成的多样性方面具有优势，但可能在生成的场景物理合规性和逻辑一致性上存在一定不足，尤其是在处理复杂交通情境时。选择这两种基线的目的是为了从不同角度评估本研究方法在生成场景质量、效率以及对复杂和模糊指令的鲁棒性方面的优势。

(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，能够高效处理复杂的计算任务，确保场景生成的实时性和响应速度。此外，配备的专业级显卡为图形处理和并行计算任务提供了强大支持，尤其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），确保数据处理和存储高效、稳定。
评估指标：为了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：该指标衡量生成一个场景所需的时间，反映了系统的效率。较短的生成时间有助于提高测试的灵活性和响应速度。
场景物理合规性：评估生成场景是否符合物理约束条件，如车辆的最大加速度、最小转弯半径等。物理合规性确保了场景的真实性，生成的场景能真实反映自动驾驶系统的实际驾驶条件。
场景逻辑一致性：检查场景中各元素之间的逻辑关系是否合理，确保生成的场景符合法律规定的交通规则。逻辑一致性是验证场景真实性和可接受性的核心标准。
场景多样性：统计生成场景的类型和数量，衡量系统的多样性和生成能力。多样性高的系统能为自动驾驶系统提供更广泛的测试场景，从而有效覆盖更多潜在的驾驶情况。
碰撞率：在仿真环境中运行生成的场景，并统计发生碰撞的频率。碰撞率反映了生成场景的安全性，较低的碰撞率意味着生成的场景

能够较好地模拟真实世界中的安全驾驶环境。

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。

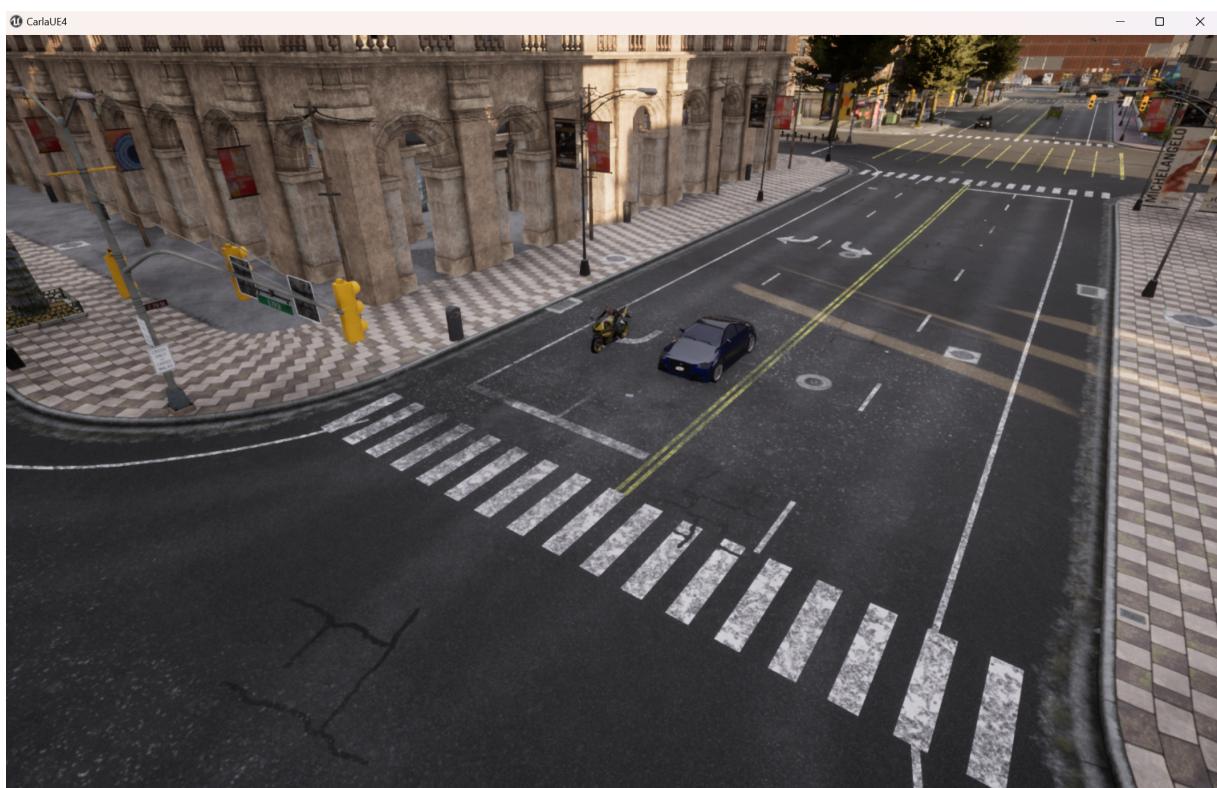


图 4-1 摩托车与汽车在红绿灯前等待信号的场景截图

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

4.3 实验总结

本章着重描述了基于自然语言生成三维交通场景的核心流程：“语言描述→场景代码生成→仿真构建”。此流程展示了如何通过自然语言输入、自动脚本生成与仿真执行的技术路径，实现复杂三维交通场景的高效构建。

- **语言描述的输入与解析:**整个流程的第一步是通过 `scenario_descriptions.txt` 文件获取自然语言描述。每条描述代表了一个特定的交通场景，包括交通参与者的类型、数量、初始位置、行为意图等信息。用户通过直观的语言输入定义了所需的场景特征。系统通过 `retrieve.py` 脚本对输入的自然语言进行语义解析。解析的目标是从文本中识别出场景的关键元素，比如车辆的种类、行驶方向、速度，甚至是动态行为（如超车、停车、避让等）。该步骤主要依赖自然语言处理（NLP）技术，通过文本特征提取与词汇映射，将模糊的语言信息转换为有结构的场景要素。
- **场景代码的自动生成:**一旦自然语言描述被解析，系统通过预设的规则与模板生成符合 Scenic 语法规范的场景脚本。Scenic 是一种针对三维仿真场景描述的语言，其特点是灵活且适用于动态场景建模。系统根据输入的描述生成的代码包含了场景中参与者的初始化设置（如车辆的位置、速度、运动轨迹等）以及可能的动态行为（如交通参与者的交互、变道、超车等）。生成的 Scenic 脚本是可执行的，并能传递给 Scenic 编译器作为输入文件。此阶段的自动化程度高，能够在大部分场景下自动生成相应的场景代码，减少了人工干预的需要。

- **仿真场景的构建与执行:**生成的 Scenic 场景脚本会被传递到仿真平台,主要是 CARLA,通过 Scenic-CARLA 接口执行。CARLA 是一个开源的自动驾驶仿真平台,它能够根据场景脚本中的设置构建出真实感强、动态可控的三维仿真场景。在此过程中,系统会依照 Scenic 脚本中的配置加载地图、部署交通参与者(如汽车、行人等)、设置场景初始状态(如天气、时间、环境条件等)。此外,仿真平台还根据脚本中的动态行为设定实时执行参与者的运动轨迹与交互。例如,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,确保生成的仿真场景在空间和行为上都符合预期。
- **系统流程特性与优势:**整个“语言描述 → 场景代码生成 → 仿真构建”的流程展现了系统的自动化处理能力,尤其是自然语言到仿真场景代码生成的桥接。通过 `retrieve.py` 脚本,用户能够快速通过文本输入生成三维交通场景,减少了手动构建仿真场景的工作量。此外,该流程支持生成复杂场景和动态交互行为,能够精准还原交通系统中的各种交互情境,如追尾、避让、变道等,为自动驾驶系统的测试与评估提供了有效的支持。

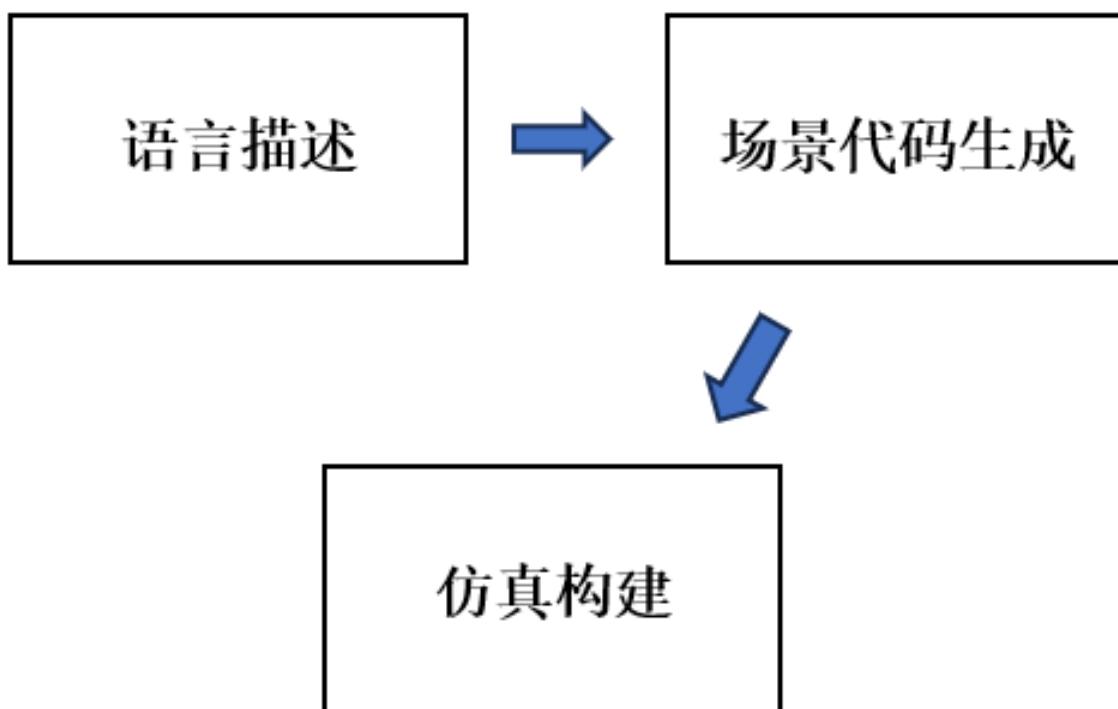


图 4-5 从自然语言描述到仿真场景构建的流程图

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。

通过日志记录与脚本打点方式采集各阶段耗时，并以平均响应时长 (ms)、方差等指标量化性能表现。此外，系统在多轮连续输入下的响应稳定性亦被纳入评估。

5.2 实验设置

硬件配置：处理器：Intel Core i9、显卡：3060、内存：32GB

软件环境：虚拟环境：项目中使用了两个虚拟环境

1. carla (Python 3.7)：用于运行 CARLA 0.9.15 版本进行仿真和场景生成。
2. chatscene (Python 3.8)：用于处理与生成自然语言描述相关的任务，以及进行场景量化评估。

依赖库：安装了必要的依赖，包括但不限于 openai、sentence_transformers、torch 等。

输入数据：本实验使用约 20 条自然语言描述作为输入样本，用于生成自动驾驶场景。这些描述涵盖了多种交通情境与突发事件，例如：自行车在红绿灯前等待信号、行人突然横穿街道、道路上突现障碍物等。

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：

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

评分区间	场景占比
5分	62%
4分	25%
3分及以下	13%

图 5-1 语义保真度评估结果

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\adv_train\train_scenario\sa
>> Saving evaluation records to C:\Users\24448\Desktop\ChatScene\scripts\log\adv_train\train_scenario\sa
>> Begin to run the scene...
```

图 5-2 准确率分析

```
avg_time_spent        0.0
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\adv_train\train_scenario\sa
Saving evaluation records to C:\Users\24448\Desktop\ChatScene\scripts\log\adv_train\train_scenario\sa
Begin to run the scene...
```

图 5-3 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：

为验证系统在实际使用场景中的运行效率，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进行了测量与分析。系统整体生成流程的平均时间如下所示：

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：1.4s
- **总耗时：平均 2.9s/条输入**

图 5-4 系统生成过程时间分析

通过上述统计可以看出，本系统在标准硬件配置下可将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时，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况下，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。

总体而言，系统具备良好的响应效率和可扩展性，不仅支持单条交互式输入，也能满足批量生成需求，适用于大规模自动驾驶测试、仿真训练、或仿真场景库构建等实际应用任务。如需进一步提升效率，可通过以下途径优化：

- 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

随着自动驾驶技术的迅速发展，如何在复杂的交通环境中通过高效的场景生成与感知系统提升自动驾驶系统的智能性和安全性成为了研究的关键。本文通过设计和实现了一种基于自然语言处理的自动驾驶仿真场景生成方法，并结合 Carla 仿真平台，探讨了如何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的决策能力。

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，采用了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了场景生成的灵活性和多样性。通过对输入的自然语言描述进行语义分析，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。此外，本文还设计了一个集成的感知系统，通过深度学习与图像处理技术实现了对交通目标的检测、跟踪与行为预测，为后续的决策与控制提供支持。

实验结果表明，基于自然语言生成的仿真场景能够准确反映输入描述的语义，并通过感知系统实时跟踪和预测交通目标的行为意图。系统能够在复杂的交通环境中进行稳定运行，验证了自然语言描述与自动驾驶场景生成的可行性和有效性。

本文的贡献在于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，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。研究中所采用的 Carla 仿真平台为系统的验证与优化提供了丰富的测试数据，未来的研究成果有望为智能驾驶系统的多样化场景生成和感知能力的提升提供更多的技术支持。

6.2 研究不足

尽管本文在自然语言生成场景与感知系统的研究中取得了一定的进展，但在一些方面仍存在不足与局限。首先，当前的场景生成方法虽然能够生成基本的交通场景，但在处理更为复杂和动态的环境（如极端天气、特殊道路条件等）时，仍显得力不从心。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仍有待提升，特别是在面对不常见的交通情境时，生成的场景可能缺乏足够的丰富性和真实感。

其次，尽管本文的感知系统能够在大多数情况下准确跟踪和预测目标行为，但在高密度交通或目标交错的情况下，系统可能会出现目标跟踪的失误，影响行为预测的准确性。在多目标竞赛和复杂背景下，现有的目标跟踪算法仍可能受到影响，导致系统在长期跟踪时出现问题。

此外，尽管本研究结合了深度学习和物理建模来进行行为意图分析，但现有方法依赖于物理模型的简单规则，可能难以捕捉更为复杂和多变的驾驶行为，尤其是当多个目标之间存在复杂交互时，系统可能无法准确预测其行为。

最后，系统的实时性和计算性能在处理高分辨率图像和复杂场景时仍存在一定瓶

颈。虽然现有系统能够满足基本的实时性要求，但在高负载条件下，响应时间和处理速度可能受到影响，这在实际应用中可能带来挑战。

6.3 后续优化方向

在未来的研究中，系统的优化将集中在以下几个方面：

首先，提升场景生成的多样性和复杂度将是未来研究的重点。当前系统生成的交通场景主要以基本场景为主，面对复杂交通情况时仍然存在不足。未来的研究将采用更先进的自然语言处理和场景生成模型，结合强化学习等方法，提高系统在极端天气、事故等复杂情况中的应对能力，生成更加多样化和真实的场景。

其次，目标跟踪与行为意图预测的精度将进一步提升。虽然现有的跟踪算法在大多数情况下能够保持目标稳定跟踪，但在高密度或复杂背景下，仍存在一定的精度问题。未来的工作将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，意图预测将结合更多的情境因素，如交通规则和社会行为，以提升对复杂驾驶行为的预测精度。

再者，系统的实时性与计算性能将持续优化。随着场景复杂度的增加，系统的实时性和处理能力将成为关键问题。未来的研究将重点研究高效的图像处理和计算方法，利用硬件加速与算法优化，提高系统的计算效率，确保在高负载和复杂场景下依旧能够保持流畅的运行。

最后，未来的研究将结合真实道路测试与仿真验证，将研究成果转化为实际应用。通过在真实环境中的测试，进一步评估系统在实际道路条件下的表现，确保系统在多样化和复杂的交通环境中的稳定性与可靠性。

通过以上优化，未来的系统将能够更好地应对动态复杂的交通场景，提高自动驾驶系统的智能感知与决策能力，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。

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

1. 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

# no need for faiss currently
# import faiss

# Argument parsing
parser = argparse.ArgumentParser(description="Set up configuration")
parser.add_argument('--port_ip', type=int, default=2000, help='Port IP')
parser.add_argument('--topk', type=int, default=3, help='Top K values')
parser.add_argument('--model', type=str, default='gpt-4o', help="Model name (default: 'gpt-4o'), also support transformer")
parser.add_argument('--use_llm', action='store_true', help='if use llm')
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', 'prompt'))
behavior_prompt = load_file(osp.join(local_path, 'retrieve', 'prompt'))
geometry_prompt = load_file(osp.join(local_path, 'retrieve', 'prompt'))
spawn_prompt = load_file(osp.join(local_path, 'retrieve', 'prompt'))
scenario_descriptions = load_file(osp.join(local_path, 'retrieve', 'prompt'))

# □ 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}")

word_embedding_model = models.Transformer(model_dir, max_seq_length=128)
pooling_model = models.Pooling(
    word_embedding_model.get_word_embedding_dimension(),
    pooling_mode='mean'
)
encoder = SentenceTransformer(modules=[word_embedding_model, pooling_model])
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')
geometry_embeddings = encoder.encode(geometry_descriptions, device='cuda')
spawn_embeddings = encoder.encode(spawn_descriptions, device='cuda')

# This is the head for scenic file, you can modify the carla map or environment
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', 'scenariolog.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'])

# 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:(.*?)" , response)
        if not match:
            raise ValueError("Failed to extract components from the response")
    except Exception as e:
        print(f"Error processing scenario {q}: {e}")

    log_writer.writerow([current_scenario, match.group(1), match.group(2)])
```

```
current_adv_object, current_behavior, current_geometry, current_spawn

# Retrieve the top K relevant snippets
top_behavior_descriptions, top_behavior_snippets = retrieve_topk(
    top_behavior_descriptions, top_behavior_snippets = retrieve_topk(
        top_geometry_descriptions, top_geometry_snippets = retrieve_topk(
            top_spawn_descriptions, top_spawn_snippets = retrieve_topk(encode

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

generated_geometry_code = generate_code_snippet(
    llm_model, geometry_prompt, top_geometry_descriptions, top_geometry_snippets)

generated_spawn_code = generate_code_snippet(
    llm_model, spawn_prompt, top_spawn_descriptions, top_spawn_snippets)

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

Town, generated_geometry_code = generated_geometry_code.split('\n')
scenic_code = '\n'.join([f"'''{current_scenario}'''", Town, head,
    save_scenic_code(local_path, port_ip, scenic_code, q)

except Exception as e:
    log_writer.writerow([current_scenario, '', '', '', '', '', '', '',
        print(f"Failure for scenario: {current_scenario} - Error: {e}")])
```

场景合成与仿真模块

1. 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_variables
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('..'))

    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=['eval', 'train'])
    parser.add_argument('--agent_cfg', nargs='*', type=str, default='cfgs/agents/agent.cfg')
    parser.add_argument('--scenario_cfg', nargs='*', type=str, default='cfgs/scenarios/scenario.cfg')
    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)
    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='frame skip')
    parser.add_argument('--port', type=int, default=2002, help='port')
```

```
parser.add_argument('--tm_port', type=int, default=8002, help='tm port')
parser.add_argument('--fixed_delta_seconds', type=float, default=0.1, help='fixed delta seconds')
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_config = load_config(agent_config_path)

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

        agent_config['load_dir'] = osp.join(agent_config['load_dir'], 'adv_train')
        # 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)
        scenario_config.update(args_dict)
        scenario_config['num_scenario'] = 1 #### 'the num_scenario can only be 1'
        runner = ScenicRunner(agent_config, scenario_config)

        # start running
        runner.run()
```

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

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

1. 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_des'
HISTORY_FILE = 'D:/sceneMain/chatScene/retrieve/scenario_history'
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调整

    # 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')
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_sce

    # 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()
```

2. 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_variables
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(os.path.dirname(__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=['eval', 'train'])
    parser.add_argument('--agent_cfg', nargs='*', type=str, default='agents/agent.cfg')
    parser.add_argument('--scenario_cfg', nargs='*', type=str, default='scenarios/scenario.cfg')
    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)
    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='frame skip')
    parser.add_argument('--port', type=int, default=2002, help='port')
    parser.add_argument('--tm_port', type=int, default=8002, help='tm port')
    parser.add_argument('--fixed_delta_seconds', type=float, default=0.05, help='fixed delta seconds')
```

```
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/con')
        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'], "d"

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

        args.output_dir = osp.join('log', 'adv_train', args.mode, agent_c
        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 b
        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])
```

其他

1. 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_variables
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(os.path.dirname(__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', 'eval', 'test'], help='Training mode')
parser.add_argument('--agent_cfg', nargs='*', type=str, default=[], help='Path to agent configuration file')
parser.add_argument('--scenario_cfg', nargs='*', type=str, default=[], help='Path to scenario configuration file')
parser.add_argument('--continue_agent_training', '-cat', type=bool, default=False, help='Continue training the agent')
parser.add_argument('--continue_scenario_training', '-cst', type=bool, default=False, help='Continue training the scenario')

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='Number of scenarios to generate')
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='Frame skip value')
parser.add_argument('--port', type=int, default=2002, help='Port number for communication')
parser.add_argument('--tm_port', type=int, default=8002, help='Port number for time synchronization')
parser.add_argument('--fixed_delta_seconds', type=float, default=0.03333333333333333, help='Fixed delta seconds for simulation')
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"")
# 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,
                                    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
    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])
```

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(os.path.dirname(__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=['eval', 'train'])
    parser.add_argument('--agent_cfg', nargs='*', type=str, default=[None])
    parser.add_argument('--scenario_cfg', nargs='*', type=str, default=[None])
    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)
    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='frame skip')
    parser.add_argument('--port', type=int, default=2002, help='port to connect to')
    parser.add_argument('--tm_port', type=int, default=8002, help='port to receive tm data')
    parser.add_argument('--fixed_delta_seconds', type=float, default=0.05)
    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_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}_{args.exp_name}')

# load scenario config
scenario_config_path = osp.join(args.ROOT_DIR, 'safebench/scenarios')
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'])
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 1'
    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])
```

1. config.py

1. config.py

```
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 all)')
argparser.add_argument(
    '-x', '--xodr-path',
    metavar='XODR_FILE_PATH',
    help='load a new map with a minimum physical road representation of the area')
argparser.add_argument(
    '--osm-path',
    metavar='OSM_FILE_PATH',
    help='load a new map with a minimum physical road representation of the area')
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 None  
  
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 %r 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)
```