From Road to Code: Neuro-Symbolic Program Synthesis for Autonomous Driving Scene Translation and Analysis

Johnathan Leung JLEUNG18@CS.UNC.EDU
Guansen Tong GTONG@CS.UNC.EDU
Parasara Sridhar Duggirala PSD@CS.UNC.EDU
Praneeth Chakravarthula CPK@CS.UNC.EDU
Department of Computer Science
The University of North Carolina at Chapel Hill
Chapel Hill, NC 27514, USA

Abstract

Translating real-world scenarios into simulation environments is essential for the safe, cost-effective, and scalable development of autonomous vehicles. Simulations enable rigorous testing of complex, rare, and hazardous scenarios, while also allowing for rapid iteration, data generation, and exposure to diverse conditions. However, the real-to-sim gap remains a significant challenge, as automated methods often fail to accurately capture real-world conditions, and manual scenario generation is labor-intensive and struggles to replicate realistic dynamics and unpredictable human behavior.

In this work, we propose **Road2Code**, a framework that bridges the gap between real-world traffic data and simulation by leveraging neuro-symbolic program synthesis. Road2Code translates real-world driving scenarios into Scenic programs[1] for the CARLA simulator[2], utilizing large language models for code generation. To enhance efficiency, we employ a distillation approach, where a large language teacher model generates reasoning processes that refine training for a smaller student model used for inference. Road2Code enhances simulation fidelity by accurately modeling real-world scenarios and agent behaviors while enabling scenario editing and counterfactual analysis, providing essential tools for testing and refining autonomous vehicle behavior. This direct link between real-world data and simulation lays a foundation for advancing trustworthy and transparent autonomous driving research, accelerating progress toward reliable autonomous vehicle systems.

Keywords: Neuro-symbolic Programming, Large Language Models, Artificial Intelligence, Autonomous Driving.

1. Introduction

Simulating autonomous driving scenarios is essential for autonomous vehicle (AV) systems development as it is less costly, time-consuming, and limited in scope, compared to real-world testing Ljungbergh et al. (2025) and it is easy to test "edge cases" such as sudden pedestrian crossings or unexpected vehicle approaches which are otherwise difficult to test Kalra and Paddock (2016). Simulation provides a risk-free, scalable environment for AV testing to enable iterative improvement of perception and planning Rong et al. (2020), accelerate training Chen et al. (2020), enable dynamic adjustments in vehicle behavior Filos et al. (2020), and support rigorous validation and verification Li et al. (2023).

However, existing frameworks often fall short in capturing the complexity of real-world driving and traffic. Most frameworks rely heavily on pre-constructed, deterministic scenarios

[1] Scenic is a domain-specific probabilistic language for interpretable traffic scenario generation
[2] CARLA is an open-source simulator for autonomous driving, for testing self-driving systems

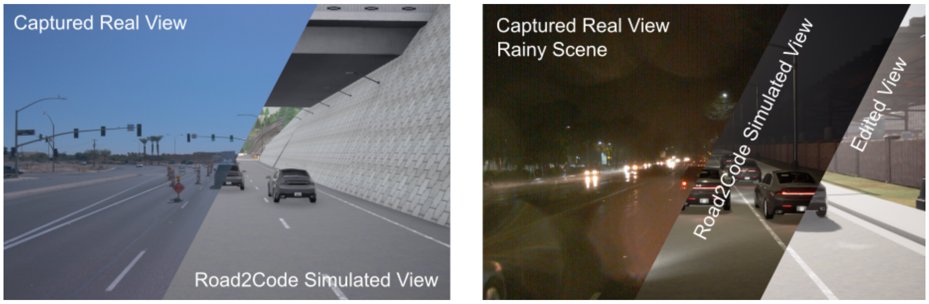


Figure 1: Road2Code converts real-world video into realistic CARLA simulations using Scenic, a domain-specific language. It preserves key elements like road structure, vehicle positions, and agent behavior (left). It also enables scene editing for further analysis and scenario refinement (right).

and hand-coded agent behavior models that lack the realism and unpredictability inherent in actual traffic Chao et al. (2018a,b). This limits their ability to effectively represent critical and nuanced events necessary for robust AV testing. Moreover, traditional simulation methods struggle to seamlessly integrate real-world sensory input, into simulation, further reducing their fidelity and practical relevance Chao et al. (2018a); Li et al. (2019). Addressing these limitations requires a *real-to-simulation* framework that can automatically translate real-world driving observations into realistic, editable simulation scenarios, providing AV systems with comprehensive exposure to the full spectrum of driving situations they may encounter on the road.

In this paper, we take the first steps toward bridging the substantial gap between real-world traffic and simulated driving environments, aiming to create adaptable, high-fidelity simulations that reflect the complexity of real-world traffic while remaining easily editable and interpretable. To achieve this, we propose a neuro-symbolic approach that generates simulated scenes directly from real-world video inputs, enabling seamless integration of real-world data into simulation frameworks. Using the powerful code generation and reasoning capabilities of Large Language Models (LLMs) Achiam et al. (2023); Roziere et al. (2023); Touvron et al. (2023); Anil et al. (2023); Devlin et al. (2019), we extract vehicle trajectories from the input video data and translate their relative motions into Scenic code Fremont et al. (2019) using our neuro-symbolic synthesis framework. This code can then be loaded into simulators such as CARLA Dosovitskiy et al. (2017) for subsequent testing and analysis.

Large models like the GPT family Achiam et al. (2023); OpenAI (2024) excel at code generation but require billions of parameters, demanding significant computational and memory resources Xu et al. (2024). Therefore, we employ a smaller language model for program generation by distilling knowledge from a teacher model. Using Zero-Shot Chain-of-Thought (ZS-CoT) prompting Kojima et al. (2023), the teacher model generates reasoning steps that link input scenarios to code, which are then incorporated into the student model for fine-tuning and inference, improving efficiency without sacrificing performance.

In summary, our key contributions in this work are:

- We introduce **Road2Code**, a framework that translates real-world driving scenarios as captured by cameras and LiDAR sensors into symbolic representations. Road2Code models diverse traffic patterns and vehicle behaviors, as shown in Figure 1, making it well-suited for autonomous vehicle certification and testing.

- We harness the reasoning capabilities of Large Language Models for program generation, employing a Zero-shot Chain-of-Thought prompting approach to guide the program synthesizer in generating accurate and interpretable neuro-symbolic code that captures agent movements and behaviors in real traffic scenarios.
- We demonstrate that scenarios generated from real-world videos are easily editable within our framework (for example, Figure 1, right). Specifically, applications such as scene translation, editing, and post-mortem analysis highlight Road2Code's utility for autonomous driving simulations and comprehensive vehicle behavior testing prior to deployment.

2. Related Work

Neuro-symbolic Program Synthesis. Program synthesis—generating programs from high-level task specifications—has long been a challenge in computer science Biermann (1978); Summers (1977). Traditional approaches to program synthesis rely on automated search and reasoning but are limited by engineering complexity and scalability Parisotto et al. (2016). Neuro-symbolic methods, which combine deep learning with symbolic reasoning, have emerged as a promising alternative Chaudhuri et al. (2021); Devlin et al. (2017); Chen et al. (2021b); Hsu et al. (2023); Okamoto and Parmar (2024); Dang-Nhu (2020); Mao et al. (2019); Stammer et al. (2021). These methods leverage deep learning for processing unstructured data while using symbolic representations for logical reasoning, interpretability, and generalization Parisotto et al. (2016); Chaudhuri et al. (2021); Jha et al. (2023). Applications of neuro-symbolic methods span textual reasoning Devlin et al. (2017), query understanding Chen et al. (2021b); Barceló et al. (2023), vision and graphics Hsu et al. (2023); Ellis et al. (2018), and multi-modal learning Mao et al. (2019); Stammer et al. (2021). In autonomous driving and robotics, neuro-symbolic programming has enabled better decision-making for autonomous agents Sun et al. (2021); Namasivayam et al. (2023); Bennajeh et al. (2019); Elmaaroufi et al. (2024a). More recently, a mixture of experts model has been used to synthesize autonomous vehicle scenarios from natural language description Elmaaroufi et al. (2024b).

Large Language Models. Recent advancements in large language models (LLMs) such as GPT-3 Achiam et al. (2023), GPT-4 Brown (2020), Llama Touvron et al. (2023), PaLM Anil et al. (2023), and BERT Devlin et al. (2019) have demonstrated strong capabilities in natural language generation Roziere et al. (2023), symbolic reasoning Chen et al. (2021a), and mathematical problem-solving Hendrycks et al. (2021). However, enhancing and adapting LLMs' reasoning for specific tasks remains a challenge. Techniques such as Chain-of-Thought (CoT) prompting Wei et al. (2022) and Zero-Shot Chain-of-Thought Kojima et al. (2023) enhance reasoning by generating intermediate steps, making models more interpretable and adaptable. Our approach leverages Zero-shot CoT to generate reasoning processes, which can enhance program synthesis abilities for simulation scenarios. A key challenge to harnessing this reasoning ability is deploying LLMs with limited computational resources. Knowledge distillation and pruning techniques Sanh et al. (2020); Muralidharan et al. (2024); Men et al. (2024); Xia et al. (2023) reduce model size while retaining performance, but typically require training a new model from from scratch. Instead, we distill the reasoning process by knowledge transfer from a teacher LLM to a lightweight student model, enabling efficient program synthesis for simulations. Training LLMs by utilizing a teacher-student for knowledge transfer have been shown to enhance LLM reasoning capability Saha et al. (2023); Ho et al. (2022).

Scene Representations and Neural Rendering. Recent advancements in 3D scene reconstruction and neural rendering, including Neural Radiance Fields (NeRF) Mildenhall et al. (2021); Tancik et al. (2022); Xu et al. (2022), Gaussian Splatting Wu et al. (2024); Kulhanek et al. (2024), and implicit representations Sitzmann et al. (2019); Chen and Zhang (2019); Park et al. (2019), have significantly improved autonomous driving simulations by enabling novel view synthesis and sensor data generation. While these methods can render both static and dynamic scenes Pumarola et al. (2021); Gao et al. (2021), they lack compositionality, making it difficult to edit individual scene elements—an essential requirement for flexible scenario testing in AV simulations. Recent efforts Ost et al. (2021); Tonderski et al. (2024); Yang et al. (2023); Khan et al. (2024); Bashetty et al. (2020) have introduced editable scene representations, but they still fall short of providing programmatic control over complex driving scenarios. In contrast, by representing traffic scenes symbolically, our Road2Code approach enables precise control, scenario editing, and counterfactual analysis—capabilities that neural rendering lacks. The work that is closest to us in recreating real-world scenarios is Miao et al. (2024) however the main approach is fundamentally different. Our approaches uses model distillation and fine-tuning of foundation models whereas Miao et al. (2024) uses prompt engineering.

3. Road2Code Neuro-Symbolic Synthesis

3.1. Problem Formulation

Generating realistic and editable autonomous driving scenarios require *structured programmatic representations* that accurately reflect real-world conditions. Given an input ego-vehicle video V, our goal is to generate a Scenic program P that encodes the scene, including the road structure, agent behaviors, and dynamic interactions, which can then be rendered in CARLA for simulation.

Formally, given an input video sequence $V = \{I_t\}_{t=1}^{T}, I_t \in \mathbb{R}^{H\times W\times 3}$, where I_t is the RGB frame at time t, our goal is to generate a programmatic representation:

$$P = \{e, a\}, e \in \mathcal{R}, a = \{a_i\}_{i=1}^{N}, \tag{1}$$

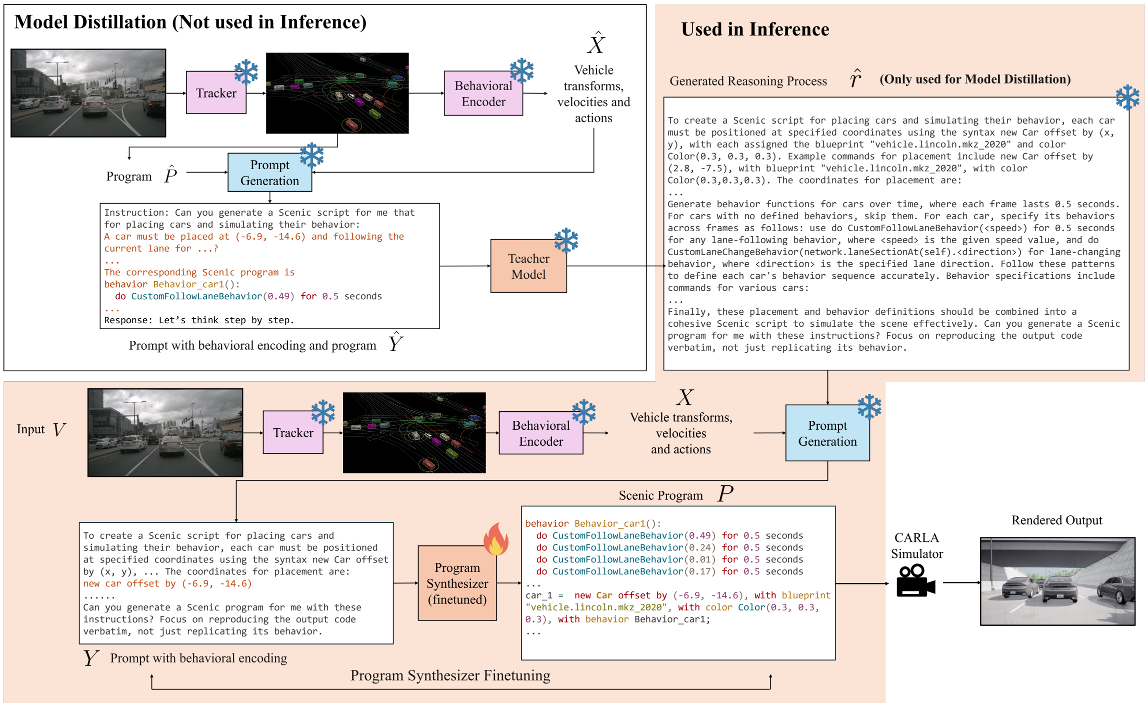
where e represents the road and environment, and a_i represents the behaviors of the ith agent (vehicle). The simulator function h then renders the scene:

$$\hat{V} = h(P), \hat{V} \approx V, \tag{2}$$

ensuring realism and fidelity between the real and simulated scene.

3.2. Road2Code Architecture

Road2Code consists of multiple processing stages, leveraging LLMs for program synthesis and neuro-symbolic reasoning for structured representation learning. We illustrate the architecture in Figure 2 and describe it here.


Model Distillation (Not used in Inference)
Tracker
Behavioral Encoder
$\hat{X}$
Vehicle transforms, velocities and actions
Program $\hat{P}$
Prompt Generation
Teacher Model
Prompt with behavioral encoding and program $\hat{Y}$
Used in Inference
Generated Reasoning Process $\hat{r}$ (Only used for Model Distillation)
Input V
Tracker
Behavioral Encoder
X
Vehicle transforms, velocities and actions
Prompt Generation
Scenic Program P
Program Synthesizer (finetuned)
Y Prompt with behavioral encoding
Program Synthesizer Finetuning
CARLA Simulator
Rendered Output

```
Instruction: Can you generate a Scenic script for me that
for placing cars and simulating their behavior:
A car must be placed at (-6.9, -14.6) and following the
current lane for ...?
...
The corresponding Scenic program is
behavior Behavior_car1():
  do CustomFollowLaneBehavior(0.49) for 0.5 seconds
...
Response: Let’s think step by step.
```

```
To create a Scenic script for placing cars and simulating their behavior, each car
must be positioned at specified coordinates using the syntax new Car offset by (x,
y), with each assigned the blueprint "vehicle.lincoln.mkz_2020" and color
Color(0.3, 0.3, 0.3). Example commands for placement include new Car offset by
(2.8, -7.5), with blueprint "vehicle.lincoln.mkz_2020", with color
Color(0.3,0.3,0.3). The coordinates for placement are:
...
Generate behavior functions for cars over time, where each frame lasts 0.5 seconds.
For cars with no defined behaviors, skip them. For each car, specify its behaviors
across frames as follows: use do CustomFollowLaneBehavior(<speed>) for 0.5 seconds
for any lane-following behavior, where <speed> is the given speed value, and do
CustomLaneChangeBehavior(network.laneSectionAt(self).<direction>) for lane-changing
behavior, where <direction> is the specified lane direction. Follow these patterns
to define each car's behavior sequence accurately. Behavior specifications include
commands for various cars:
...
Finally, these placement and behavior definitions should be combined into a
cohesive Scenic script to simulate the scene effectively. Can you generate a Scenic
program for me with these instructions? Focus on reproducing the output code
verbatim, not just replicating its behavior.
```

```
To create a Scenic script for placing cars and
simulating their behavior, each car must be positioned
at specified coordinates using the syntax new Car offset
by (x, y), ... The coordinates for placement are:
new car offset by (-6.9, -14.6)
......
Can you generate a Scenic program for me with these
instructions? Focus on reproducing the output code
verbatim, not just replicating its behavior.
```

```
behavior Behavior_car1():
  do CustomFollowLaneBehavior(0.49) for 0.5 seconds
  do CustomFollowLaneBehavior(0.24) for 0.5 seconds
  do CustomFollowLaneBehavior(0.01) for 0.5 seconds
  do CustomFollowLaneBehavior(0.17) for 0.5 seconds
...
car_1 =  new Car offset by (-6.9, -14.6), with blueprint
"vehicle.lincoln.mkz_2020", with color Color(0.3, 0.3,
0.3), with behavior Behavior_car1;
...
```

Figure 2: Road2Code extracts vehicle tracks and translates their motions into Scenic programs. Using Zero-Shot Chain-of-Thought, a teacher model generates reasoning, which is integrated into fine-tuning prompts while training the program synthesizer. At inference time, the generated program P from the video V is deployed in CARLA for evaluation.

Tracking Module T: The tracking module extracts vehicle trajectories from V, producing a set of 3D vehicle positions relative to ego:

$$X_t = \{x_{i,t}\}_{i=1}^{N}, x_{i,t} \in \mathbb{R}^3, \tag{3}$$

where $x_{i,t}$ is the position of the ith vehicle at time t. The sequence of vehicle trajectories is then represented as:

$$X = (\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_n) \tag{4}$$

where each vehicle i has a trajectory $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,T})$. We use a pre-trained Multi-Object Tracking (MOT) model Hu et al. (2019); Chiu et al. (2021) to compute X.

Behavior Encoding Module E: Each vehicle's movement is encoded into a behavior vector:

$$B = \{b_{i,t}\}_{i=1}^{N}, b_{i,t} = (v_{i,t}, a_{i,t}), \tag{5}$$

where $v_{i,t}$ is the velocity at time t, and $a_{i,t}$ is the action at t, such as lane change or braking. This module produces an encoding function: $B = E(X)$, ensuring structured representation of autonomous agent (vehicle) behavior.

Prompt Generation Module G: The behavior encoding is converted into a structured text prompt Y via the prompt generation function G: $Y = G(B)$. This prompt serves as the input to the program synthesizer. Specifically, G encodes vehicle behaviors into a structured textual format. For instance, for each vehicle i, the initial placement is represented as: “`Place car at` $x_{i,1}$”. This process is repeated for all vehicles in X_t. Subsequently, a sequence of actions is generated, for example: “`drive forward at` $v_{0,t}$ `for 0.5 seconds, then ...`”. The resulting text prompt Y_j encapsulates the full scenario. Optionally, an additional reasoning process r can be incorporated to provide structured guidance for scenario synthesis, modifying the prompt generation function to:

$$Y = G(B, r). \tag{6}$$

This approach provides explicit agent actions and ensures interpretability, allowing LLMs to infer correct programmatic rules and aiding the program synthesizer in generating realistic and logically consistent simulation scenarios.

Program Synthesizer S: The program synthesizer S, implemented as a fine-tuned LLM, generates the Scenic program:

$$P = S(Y), \tag{7}$$

where P is the programmatic representation (see Equation (1)) that includes structured definitions such as:

$$P = \{e, a\}, a = \{\texttt{define } a_i \texttt{ with position } x_{i,1} \texttt{ and velocity } v_{i,1}\}_i. \tag{8}$$

The synthesizer translates behavior into executable code, which can then be rendered in the CARLA simulator to generate realistic and interpretable scenarios.

3.3. Teacher-Student Model Distillation

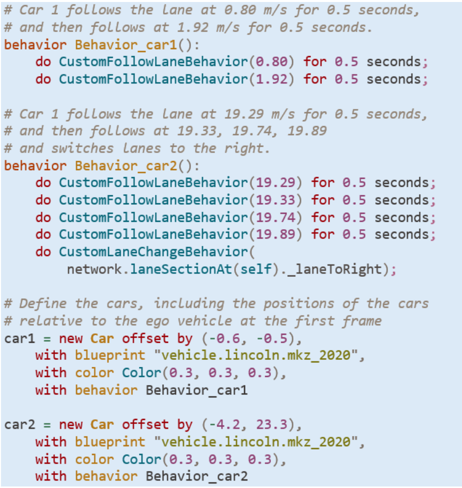
Using large language models like GPT-4o for program synthesis is computationally expensive. To reduce inference costs, we employ knowledge distillation, where a teacher model generates reasoning processes to train a smaller student model, as illustrated in Figure 2.

Teacher Model: The teacher model generates an explanation r of how scenario data X maps to the program P:

$$r = g(X, P). \tag{9}$$

Following a Zero-shot Chain-of-Thought prompting Kojima et al. (2023), we introduce structured reasoning, such as: r = “`Let's think step-by-step: Given position` $x_{i,1}$`, the vehicle must move with velocity` $v_{i,1}$”. This structured reasoning enables the student model to learn implicit relationships.

Student Model: This is fine-tuned using a dataset of pairs of coordinates and ground truth programs: $D = \{(Y_j, P_j)\}_{j=1}^{N}$, where training follows Equation (7): $\hat{P}_j = S(Y_j, r_j)$. To improve efficiency, we use QLoRA Dettmers et al. (2024) for low-rank adaptation, reducing the number of trainable parameters while retaining performance.



```
# Car 1 follows the lane at 0.80 m/s for 0.5 seconds,
# and then follows at 1.92 m/s for 0.5 seconds.
behavior Behavior_car1():
    do CustomFollowLaneBehavior(0.80) for 0.5 seconds;
    do CustomFollowLaneBehavior(1.92) for 0.5 seconds;

# Car 1 follows the lane at 19.29 m/s for 0.5 seconds,
# and then follows at 19.33, 19.74, 19.89
# and switches lanes to the right.
behavior Behavior_car2():
    do CustomFollowLaneBehavior(19.29) for 0.5 seconds;
    do CustomFollowLaneBehavior(19.33) for 0.5 seconds;
    do CustomFollowLaneBehavior(19.74) for 0.5 seconds;
    do CustomFollowLaneBehavior(19.89) for 0.5 seconds;
    do CustomLaneChangeBehavior(
        network.laneSectionAt(self)._laneToRight);

# Define the cars, including the positions of the cars
# relative to the ego vehicle at the first frame
car1 = new Car offset by (-0.6, -0.5),
    with blueprint "vehicle.lincoln.mkz_2020",
    with color Color(0.3, 0.3, 0.3),
    with behavior Behavior_car1

car2 = new Car offset by (-4.2, 23.3),
    with blueprint "vehicle.lincoln.mkz_2020",
    with color Color(0.3, 0.3, 0.3),
    with behavior Behavior_car2
```

Figure 3: This example program generated by Road2Code defines agents, initial positions, and vehicle behaviors, illustrating motion representation. Manually added comments provide clarity.

Table 1: We evaluate visual error of our model with and without Chain-of-Thought distillation. We find that distillation improves SSIM by 5.4% and reduces MSE by 47.1%, lowering errors across all metrics.

	SSIM ↑	MSE ↓	LPIPS ↓	mAP50 ↑
No Distillation	0.8079	0.1251	0.2973	0.0190
Ours with ZS-CoT Distillation	0.8515	0.0662	0.1949	0.7333

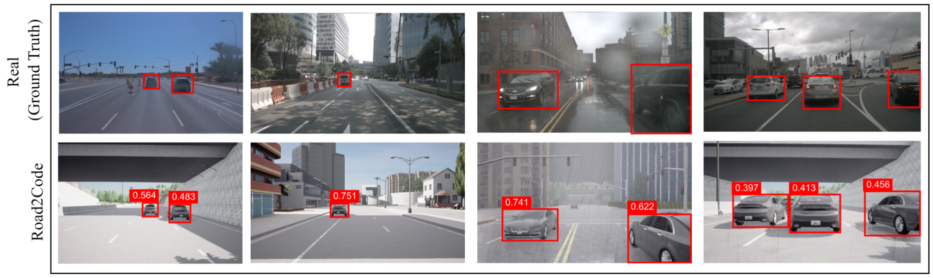


Figure 4: Real-world traffic scenes are translated to simulation using Road2Code. The simulated vehicles closely match real-world agents, with IoU scores between 0.6 and 0.7, indicating strong alignment between simulated and real bounding boxes.

3.4. Training and Inference

Training Phase: Training follows a supervised fine-tuning approach: ① extract vehicle trajectories $X_j = T(V_j)$, ② encode behavior $B_j = E(X_j)$, ③ generate prompt $Y_j = G(B_j)$ and ④ Fine-tune student model with knowledge distillation

$$\hat{P}_k = S(Y_k, g(X_k, P_k)),$$

where g generates reasoning explanations.

Inference Phase: Given an unseen driving sequence V, the interpretable scenic program is generated as:

$$P = S(G(E(T(V)))).$$

Note that *the teacher model is not required at this stage, as reasoning knowledge is already embedded in the student model.* The Scenic program can be executed in CARLA, enabling simulation, scenario editing, and counterfactual analysis.

The generated Scenic program is highly editable due to its high-level syntax, which allows users to describe agent behaviors and movements in an intuitive manner. Unlike low-level scene descriptions from 3D reconstruction-based methods Ost et al. (2021); Tonderski et al. (2024); Yang et al. (2023); Khan et al. (2024); Zhou et al. (2024), Scenic provides a structured representation that simplifies modifications.

4. Results and Discussion

4.1. Implementation

Network and Hyperparameters: We used the GPT-4o model OpenAI (2024) as the teacher model with a temperature of 1.0. For the student model, we fine-tuned a pre-trained Llama 3.1 model Touvron et al. (2023) with 8 billion parameters, quantized to 8 bits. Text generation was performed using top-k sampling ($k = 50$) with a temperature of 1.0 to balance diversity and determinism.

Training Details: The model was trained with 100 warm-up steps and 1,500 steps using the AdamW optimizer Loshchilov and Hutter (2019) at a learning rate of 0.0003. In finetuning, QLoRA hyperparameters were set to rank $r = 16$, $\alpha = 16$, and a dropout probability of 0.05. Training was conducted on a dataset of 500 prompt-program pairs generated from the nuScenes dataset. We generated the programs in the dataset from the ground truth coordinates Caesar et al. (2020).

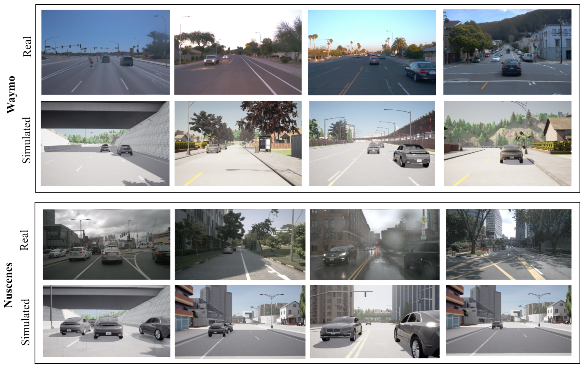


Figure 5: Scenes generated from Road2Code on instances from nuScenes and Waymo dataset.

Datasets: We evaluate our model using nuScenes Caesar et al. (2020) and Waymo Open Dataset Sun et al. (2020), large-scale datasets of real-world autonomous driving scenarios in urban environments. Waymo images have a resolution of 1920×1280 and nuScenes images are 1600×900. We extract vehicle tracks over 5 frames at 0.5-second intervals for fine-grained evaluations.

4.2. Evaluation

Qualitative Evaluation. We evaluated our model on real-world scenes from the nuScenes and Waymo datasets, with visualizations shown in Figure 5. The generated scenarios closely match the real scenes, accurately preserving vehicle placements, road layouts, and weather conditions. For instance, note that the position of the vehicles in the simulated images generated by the Road2Code framework visually match their real-world counterparts. Our framework can to handle multi-lane scenarios, placing the vehicles in the correct lane, as shown first and third scene of Waymo, and third scene in nuScenes in Figure 5. Most notably, as demonstrated in the third column of nuScenes in Figure 5, our framework can enable translating real scenes of inclement weather (rain in this case) into clear day simulations, facilitating scene analysis.

Quantitative Evaluation. We evaluate the visual similarity of our model quantitatively in a Synthetic-to-Synthetic scenario to isolate key performance factors under controlled conditions. Evaluating per-pixel quantitative performance on real-world scenes is challenging due to the lack of accurate ground truth labels. To address this, we generate eight simulated 3D scenes in Carla, varying vehicle locations, environments, and weather conditions, and use them as ground truth. These scenes are processed through our framework, and the reconstructed 3D scenes are rendered in CARLA. To assess similarity, we compute mean-squared error (MSE) as a per-pixel error metric, and SSIM Wang et al. (2004) and LPIPS Zhang et al. (2018) scores to measure perceptual fidelity.

Additionally, we evaluate bounding box accuracy on real-world scenes, by computing the bounding boxes of the agents in both ground truth and generated images using object detection, and then the mAP@0.5 score Lin et al. (2014), that is, the mean average precision where the Intersection-over-Union threshold for positive bounding box match is 0.5. As shown in Table 1, model distillation significantly improves visual similarity, generating images that more closely resemble the original scenes.

Generalization. Although our model was trained on nuScenes, it successfully generalized to novel scenes in both Waymo and nuScenes datasets, as demonstrated in Figure 4. The generated scenarios closely align with the real-world scenes, with vehicle bounding boxes exhibiting high Intersection-over-Union (IoU) scores, indicating accurate spatial correspondence between the real and synthesized environments.

4.3. Applications of Road2Code

The **Road2Code** framework unlocks new capabilities for AV testing and certification by enabling workflows that would otherwise be extremely data-intensive, impractical or infeasible. In this section, we highlight two key applications: *Scenario Editing* and *Counterfactual Analysis*, both of which are critical for designing safer AV systems and ensuring robust certification processes.

Scenario Editing — Enhancing AV Certification by Supplementing Test Scenarios: A major challenge in certification of autonomous vehicles is the sparsity of critical real-world driving scenarios. Certain high-risk situations, such as sudden pedestrian crossings, or near-miss collisions, occur rarely in real-world data. With Road2Code, we overcome this limitation by systematically modifying and generating new scenarios in a simulation environment. For instance, an AV operating in a specific neighborhood may rarely encounter a busy intersection, whereas human drivers would naturally accumulate far more experiences in such locations. Road2Code enables the targeted creation of these rare but crucial scenarios to ensure comprehensive evaluation of AV decision-making.

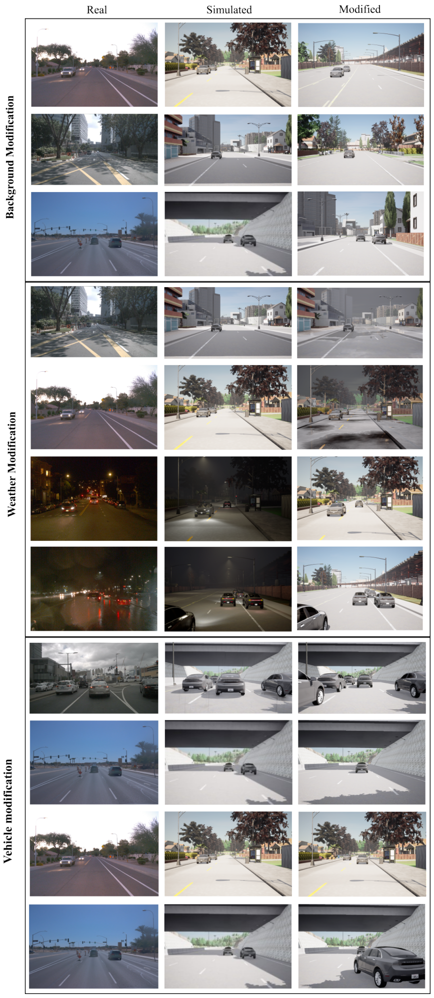


Figure 6: We modify Road2Code to generate scenes for a variety of novel unseen scenarios, adjusting environment, weather conditions, and vehicle configurations.

We first extracts a symbolic representation of a real-world scenario, then perform systematic modifications, such as adjusting road parameters (e.g., lane count, intersections), altering physical surroundings (e.g., vegetation, infrastructure elements) or introducing dynamic elements (e.g., vehicles, pedestrians, or weather conditions). These modifications allow for stress-testing AV models under a broad range of conditions without requiring extensive real-world data collection. Figure 6 demonstrates how Road2Code enables scenario augmentation by introducing additional vehicles or modifying the behaviors of existing agents, expanding the test coverage of AVs.

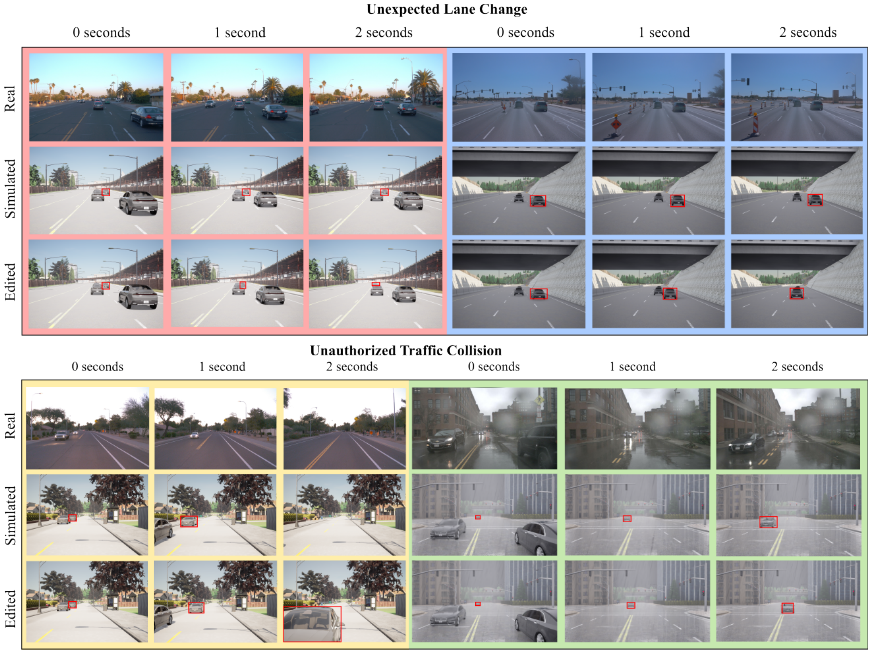


Figure 7: The Scenic program generated from Road2Code is highly editable. We showcase unexpected and unauthorized lane changes (top row) and oncoming traffic collisions (bottom). The bottom-right scenario is from nuScenes and others from Waymo dataset.

Counterfactual Analysis A fundamental tool for analyzing AV failures is *counterfactual reasoning*, which explores "what-if" scenarios by modifying past situations to investigate alternative outcomes. This is crucial for identifying whether an AV failure stems from a logical error, perception inaccuracy, or poor decision-making.

Using Road2Code, we conduct counterfactual analysis by systematically altering agent behaviors to induce potentially unsafe conditions. As illustrated in Figure 7, Road2Code enables precise manipulation of scenarios to test common accident cases that are difficult to capture in real-world datasets. This capability potentially allows for validating AV decision-making robustness under adversarial conditions and ensuring safer real-world deployment by eliminating critical vulnerabilities in systems.

5. Conclusion

We have presented a framework for converting a real-world autonomous vehicle driving scenario into a symbolic representation using the domain-specific language Scenic. We harnessed the reasoning capabilities of large language models and model distillation to efficiently generate programs with a neurosymbolic approach. By demonstrating applications of the scenario generation for evaluating autonomous driving models under hazardous scenarios, we also demonstrate the applicability of our method for robustly simulating and testing autonomous vehicles. One key extension is to integrate more visually realistic scene representation into our system. Recent efforts Ost et al. (2021); Tonderski et al. (2024); Yang et al. (2023) based on inverse rendering have introduced editable scene representations, but still fall short of providing programmatic control over complex driving scenarios. By providing a bridge to the symbolic representation, this can augment the ability of our model for autonomous driving scenarios.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. Palm 2 technical report, 2023. URL https://arxiv.org/abs/2305.10403.

Pablo Barceló, Tamara Cucumides, Floris Geerts, Juan Reutter, and Miguel Romero. A neuro-symbolic framework for answering conjunctive queries. *arXiv preprint arXiv:2310.04598*, 2023.

Sai Krishna Bashetty, Heni Ben Amor, and Georgios Fainekos. Deepcrashtest: Turning dashcam videos into virtual crash tests for automated driving systems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11353–11360. IEEE, 2020.

Anouer Bennajeh, Slim Bechikh, Lamjed Ben Said, and Samir Aknine. Bi-level decision-making modeling for an autonomous driver agent: Application in the car-following driving behavior. *Applied Artificial Intelligence*, 33(13):1157–1178, 2019.

Alan W Biermann. The inference of regular lisp programs from examples. *IEEE transactions on Systems, Man, and Cybernetics*, 8(8):585–600, 1978.

Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

Qianwen Chao, Zhigang Deng, Jiaping Ren, Qianqian Ye, and Xiaogang Jin. Realistic data-driven traffic flow animation using texture synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 24(2):1167–1178, 2018a. doi: 10.1109/TVCG.2017.2648790.

Qianwen Chao, Zhigang Deng, Yangxi Xiao, Dunbang He, Qiguang Miao, and Xiaogang Jin. Dictionary-based fidelity measure for virtual traffic. *IEEE transactions on visualization and computer graphics*, 26(3):1490–1501, 2018b.

Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, Yisong Yue, et al. Neurosymbolic programming. *Foundations and Trends® in Programming Languages*, 7(3):158–243, 2021.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021a.

Qiaochu Chen, Aaron Lamoreaux, Xinyu Wang, Greg Durrett, Osbert Bastani, and Isil Dillig. Web question answering with neurosymbolic program synthesis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 328–343, 2021b.

Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5939–5948, 2019.

Hsu-kuang Chiu, Jie Li, Rareş Ambruş, and Jeannette Bohg. Probabilistic 3d multi-modal, multi-object tracking for autonomous driving. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 14227–14233. IEEE, 2021.

Raphaël Dang-Nhu. Plans: Neuro-symbolic program learning from videos. *Advances in Neural Information Processing Systems*, 33:22445–22455, 2020.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *International conference on machine learning*, pages 990–998. PMLR, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. *Advances in neural information processing systems*, 31, 2018.

Karim Elmaaroufi, Devan Shanker, Ana Cismaru, Marcell Vazquez-Chanlatte, Alberto Sangiovanni-Vincentelli, Matei Zaharia, and Sanjit A Seshia. Scenicnl: generating probabilistic scenario programs from natural language. *arXiv preprint arXiv:2405.03709*, 2024a.

Karim Elmaaroufi, Devan Shanker, Ana Cismaru, Marcell Vazquez-Chanlatte, Alberto Sangiovanni-Vincentelli, Matei Zaharia, and Sanjit A. Seshia. Scenicnl: Generating probabilistic scenario programs from natural language, 2024b. URL https://arxiv.org/abs/2405.03709.

Angelos Filos, Panagiotis Tigkas, Rowan McAllister, Nicholas Rhinehart, Sergey Levine, and Yarin Gal. Can autonomous vehicles identify, recover from, and adapt to distribution shifts? In *International Conference on Machine Learning*, pages 3145–3153. PMLR, 2020.

Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pages 63–78, 2019.

Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Namgyu Ho, Laura Schmid, and Se-Young Yun. Large language models are reasoning teachers. *arXiv preprint arXiv:2212.10071*, 2022.

Joy Hsu, Jiayuan Mao, and Jiajun Wu. Ns3d: Neuro-symbolic grounding of 3d objects and relations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2614–2623, 2023.

Hou-Ning Hu, Qi-Zhi Cai, Dequan Wang, Ji Lin, Min Sun, Philipp Krahenbuhl, Trevor Darrell, and Fisher Yu. Joint monocular 3d vehicle detection and tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5390–5399, 2019.

Sumit Kumar Jha, Susmit Jha, Patrick Lincoln, Nathaniel D Bastian, Alvaro Velasquez, Rickard Ewetz, and Sandeep Neema. Neuro symbolic reasoning for planning: Counterexample guided inductive synthesis using large language models and satisfiability solving. *arXiv preprint arXiv:2309.16436*, 2023.

Nidhi Kalra and Susan M Paddock. Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182–193, 2016.

Mustafa Khan, Hamidreza Fazlali, Dhruv Sharma, Tongtong Cao, Dongfeng Bai, Yuan Ren, and Bingbing Liu. Autosplat: Constrained gaussian splatting for autonomous driving scene reconstruction. *arXiv preprint arXiv:2407.02598*, 2024.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Jonas Kulhanek, Songyou Peng, Zuzana Kukelova, Marc Pollefeys, and Torsten Sattler. Wildgaussians: 3d gaussian splatting in the wild. *arXiv preprint arXiv:2407.08447*, 2024.

Changwen Li, Joseph Sifakis, Qiang Wang, Rongjie Yan, and Jian Zhang. Simulation-based validation for autonomous driving systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 842–853, 2023.

W. Li, C. W. Pan, R. Zhang, J. P. Ren, Y. X. Ma, J. Fang, F. L. Yan, Q. C. Geng, X. Y. Huang, H. J. Gong, W. W. Xu, G. P. Wang, D. Manocha, and R. G. Yang. Aads: Augmented autonomous driving simulation using data-driven algorithms. *Science Robotics*, 4(28):eaaw0863, 2019. doi: 10.1126/scirobotics.aaw0863. URL https://www.science.org/doi/abs/10.1126/scirobotics.aaw0863.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollaŕ, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

William Ljungbergh, Adam Tonderski, Joakim Johnander, Holger Caesar, Kalle Åström, Michael Felsberg, and Christoffer Petersson. Neuroncap: Photorealistic closed-loop safety testing for autonomous driving. In *European Conference on Computer Vision*, pages 161–177. Springer, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect, 2024. URL https://arxiv.org/abs/2403.03853.

Yan Miao, Georgios Fainekos, Bardh Hoxha, Hideki Okamoto, Danil Prokhorov, and Sayan Mitra. From dashcam videos to driving simulations: Stress testing automated vehicles against rare events. *arXiv preprint arXiv:2411.16027*, 2024.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation, 2024. URL https://arxiv.org/abs/2407.14679.

K Namasivayam, Himanshu Singh, Vishal Bindal, Arnav Tuli, Vishwajeet Agrawal, Rahul Jain, Parag Singla, and Rohan Paul. Learning neuro-symbolic programs for language guided robot manipulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7973–7980. IEEE, 2023.

Lauren Okamoto and Paritosh Parmar. Hierarchical neurosymbolic approach for comprehensive and explainable action quality assessment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3204–3213, 2024.

OpenAI. Gpt-4 turbo documentation, 2024. URL https://platform.openai.com/docs/models/gpt-4o. Accessed: 2024-11-15.

Julian Ost, Fahim Mannan, Nils Thuerey, Julian Knodt, and Felix Heide. Neural scene graphs for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2856–2865, 2021.

Emilio Parisotto, Abdel rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis, 2016. URL https://arxiv.org/abs/1611.01855.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021.

Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. Lgsvl simulator: A high fidelity simulator for autonomous driving. In *2020 IEEE 23rd International conference on intelligent transportation systems (ITSC)*, pages 1–6. IEEE, 2020.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Swarnadeep Saha, Peter Hase, and Mohit Bansal. Can language models teach weaker agents? teacher explanations improve students via personalization. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 62869–62891, 2023.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020. URL https://arxiv.org/abs/1910.01108.

Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019.

Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3619–3629, 2021.

Phillip D Summers. A methodology for lisp program construction from examples. *Journal of the ACM (JACM)*, 24(1):161–175, 1977.

Jiankai Sun, Hao Sun, Tian Han, and Bolei Zhou. Neuro-symbolic program search for autonomous driving decision module design. In Jens Kober, Fabio Ramos, and Claire Tomlin, editors, *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 21–30. PMLR, 16–18 Nov 2021. URL https://proceedings.mlr.press/v155/sun21a.html.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8248–8258, 2022.

Adam Tonderski, Carl Lindström, Georg Hess, William Ljungbergh, Lennart Svensson, and Christoffer Petersson. Neurad: Neural rendering for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14895–14904, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Tong Wu, Yu-Jie Yuan, Ling-Xiao Zhang, Jie Yang, Yan-Pei Cao, Ling-Qi Yan, and Lin Gao. Recent advances in 3d gaussian splatting. *Computational Visual Media*, 10(4): 613–642, 2024.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Mengwei Xu, Wangsong Yin, Dongqi Cai, Rongjie Yi, Daliang Xu, Qipeng Wang, Bingyang Wu, Yihao Zhao, Chen Yang, Shihe Wang, Qiyang Zhang, Zhenyan Lu, Li Zhang, Shangguang Wang, Yuanchun Li, Yunxin Liu, Xin Jin, and Xuanzhe Liu. A survey of resource-efficient llm and multimodal foundation models, 2024. URL https://arxiv.org/abs/2401.08092.

Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5438–5448, 2022.

Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1389–1399, 2023.

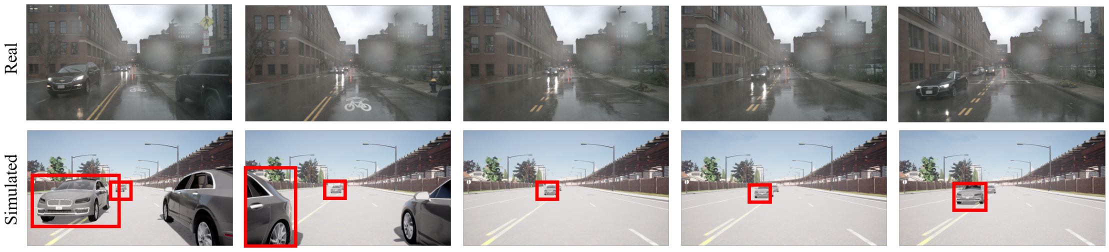
Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

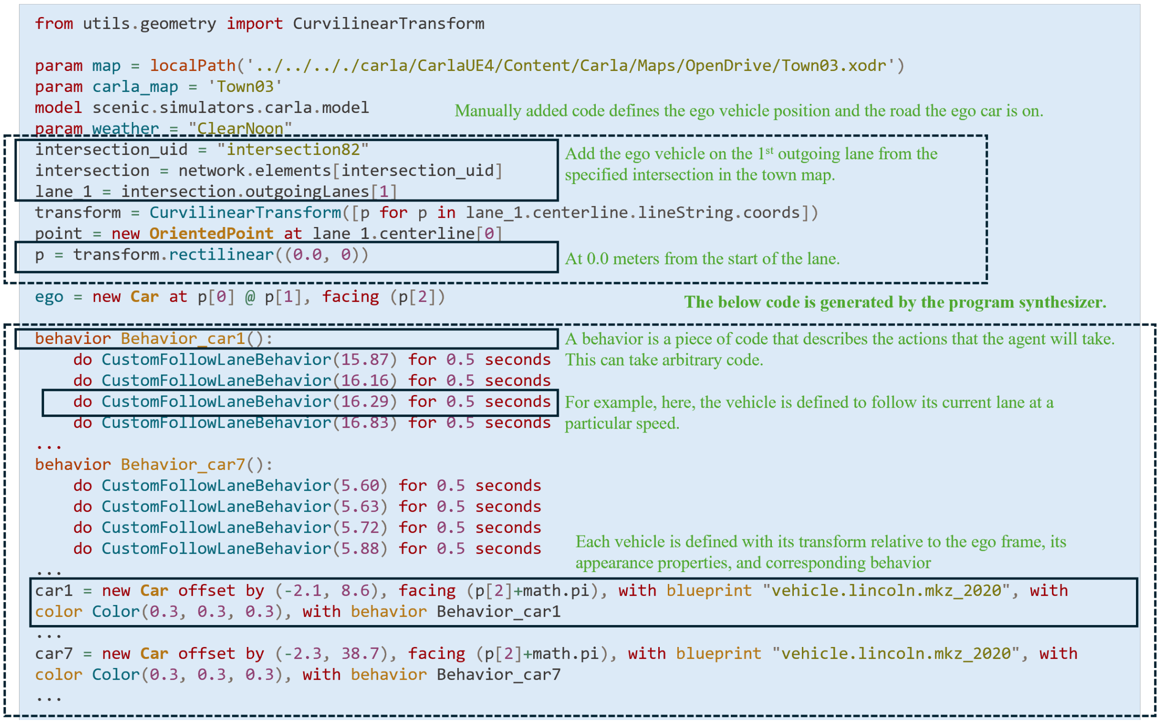
Xiaoyu Zhou, Zhiwei Lin, Xiaojun Shan, Yongtao Wang, Deqing Sun, and Ming-Hsuan Yang. Drivinggaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21634–21643, 2024.

Appendix A. Programmatic Scenario Editability

Figure 8 illustrates the structure and syntax of a Scenic program. The syntax of Scenic is a domain-specific language suited for scenario definitions in simulation-based testing. The generated program consists of a list of definitions for a list of vehicles, describing the characteristics of each vehicle. Each vehicle must correspond to a behavior to describe the motion over time. We note that the LLM has learned to generate the behavior and vehicle definition code, which is formed into a functioning program. However, other sections of the code, such as definitions of the position of the ego-vehicle, the road that the vehicle has been placed on, and essential start-up code, have not been generated by the LLM and was edited in.

We present various edits to the programs, demonstrating their ability to generate modified scenarios. This enhances the diversity of situations used in simulation and enables counterfactual testing to identify autonomous driving bugs in critical scenarios. We present some essential editing operations, which include the corresponding line in the program to edit. Examples are contained in Figures 8 to 11. We note that the aspect ratios of some images differ because we used two datasets—nuScenes, which has a wider aspect ratio than Waymo.





```
from utils.geometry import CurvilinearTransform

param map = localPath('../../../../carla/CarlaUE4/Content/Carla/Maps/OpenDrive/Town03.xodr')
param carla_map = 'Town03'
model scenic.simulators.carla.model
param weather = "ClearNoon"
intersection_uid = "intersection82"
intersection = network.elements[intersection_uid]
lane_1 = intersection.outgoingLanes[1]
transform = CurvilinearTransform([p for p in lane_1.centerline.lineString.coords])
point = new OrientedPoint at lane_1.centerline[0]
p = transform.rectilinear((0.0, 0))

ego = new Car at p[0] @ p[1], facing (p[2])

behavior Behavior_car1():
    do CustomFollowLaneBehavior(15.87) for 0.5 seconds
    do CustomFollowLaneBehavior(16.16) for 0.5 seconds
    do CustomFollowLaneBehavior(16.29) for 0.5 seconds
    do CustomFollowLaneBehavior(16.83) for 0.5 seconds
...
behavior Behavior_car7():
    do CustomFollowLaneBehavior(5.60) for 0.5 seconds
    do CustomFollowLaneBehavior(5.63) for 0.5 seconds
    do CustomFollowLaneBehavior(5.72) for 0.5 seconds
    do CustomFollowLaneBehavior(5.88) for 0.5 seconds
...
car1 = new Car offset by (-2.1, 8.6), facing (p[2]+math.pi), with blueprint "vehicle.lincoln.mkz_2020", with
color Color(0.3, 0.3, 0.3), with behavior Behavior_car1
...
car7 = new Car offset by (-2.3, 38.7), facing (p[2]+math.pi), with blueprint "vehicle.lincoln.mkz_2020", with
color Color(0.3, 0.3, 0.3), with behavior Behavior_car7
...
```

Manually added code defines the ego vehicle position and the road the ego car is on.

Add the ego vehicle on the 1st outgoing lane from the specified intersection in the town map.

At 0.0 meters from the start of the lane.

The below code is generated by the program synthesizer.

A behavior is a piece of code that describes the actions that the agent will take. This can take arbitrary code.

For example, here, the vehicle is defined to follow its current lane at a particular speed.

Each vehicle is defined with its transform relative to the ego frame, its appearance properties, and corresponding behavior

Figure 8: We present the structure of a Scenic program. A Scenic program consists of a list of specific vehicles, each with a corresponding definition. Every vehicle is paired with a behavior that describes its motion over time. The behaviors and vehicle definitions are generated by the program synthesizer.

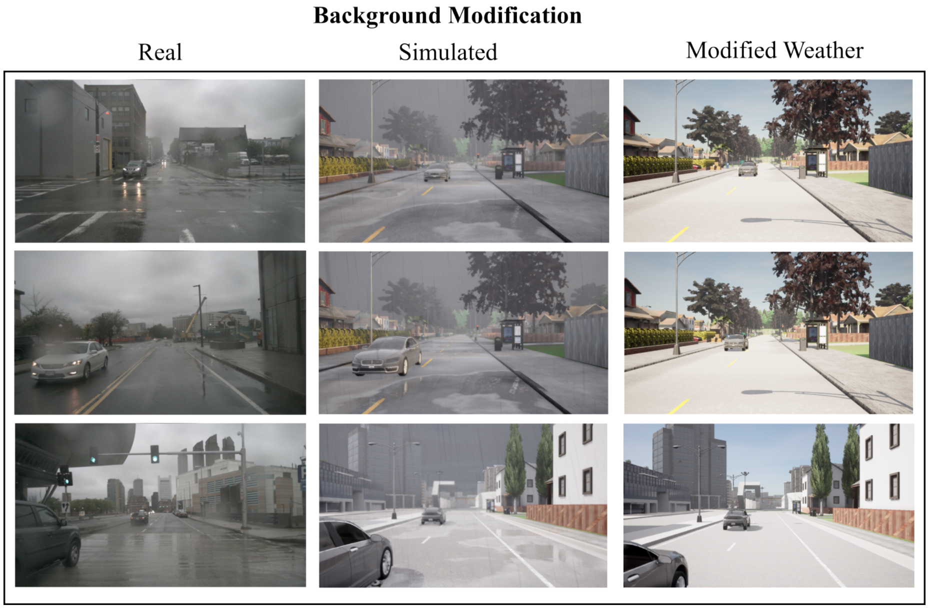
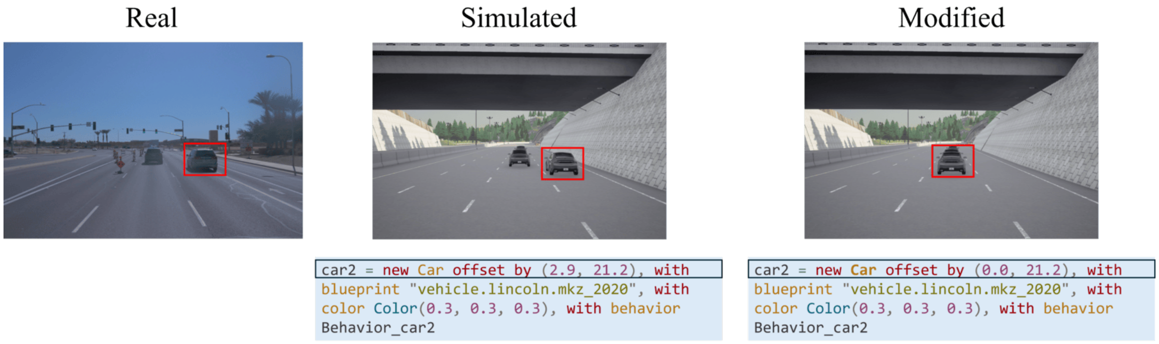


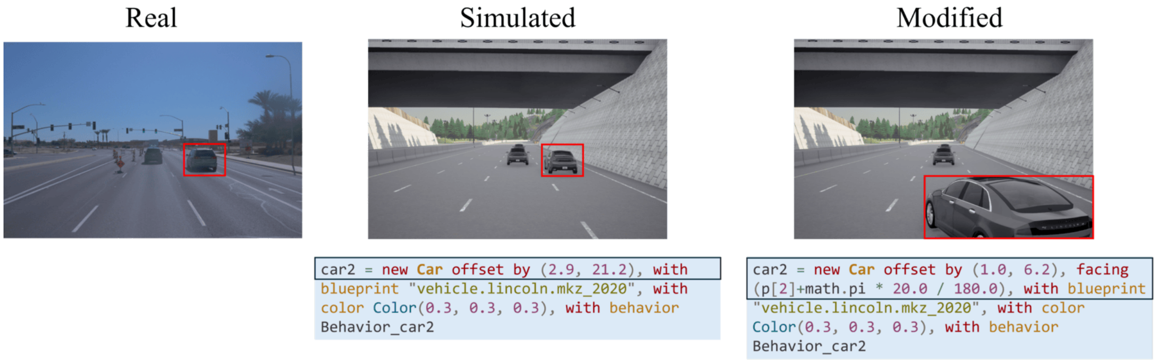
Figure 9: We present additional examples where scenario backgrounds can be manipulated; for example we can obtain night scenes and change inclement weather to render under a different weather condition.



```
car2 = new Car offset by (2.9, 21.2), with
blueprint "vehicle.lincoln.mkz_2020", with
color Color(0.3, 0.3, 0.3), with behavior
Behavior_car2
```

```
car2 = new Car offset by (0.0, 21.2), with
blueprint "vehicle.lincoln.mkz_2020", with
color Color(0.3, 0.3, 0.3), with behavior
Behavior_car2
```

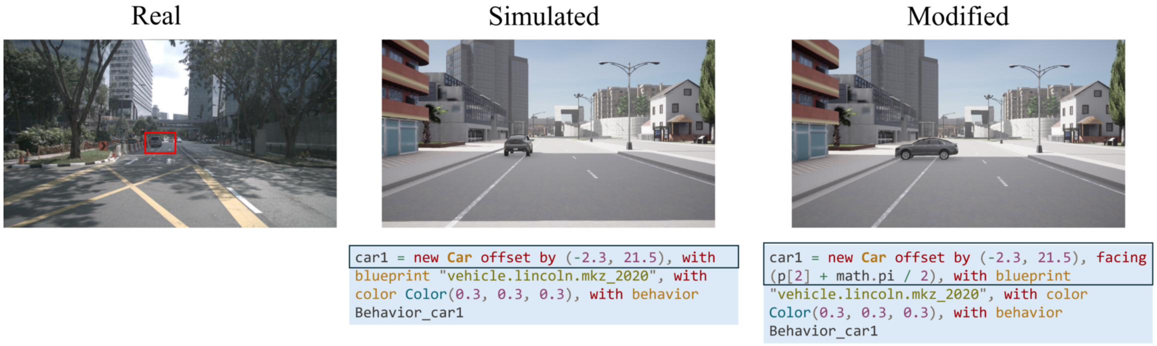
Figure 10: We can manipulate the initial position of the vehicle. The position is defined as a pair of coordinates relative to the ego vehicle frame. The x-coordinate has been shifted to 0.0, which causes the vehicle to be translated to the left lane. This shows a way to generate a new scenario to allow an autonomous driving model to handle diverse situations. The relevant modified vehicle is indicated in the illustration.



```
car2 = new Car offset by (2.9, 21.2), with
blueprint "vehicle.lincoln.mkz_2020", with
color Color(0.3, 0.3, 0.3), with behavior
Behavior_car2
```

```
car2 = new Car offset by (1.0, 6.2), facing
(p[2]+math.pi * 20.0 / 180.0), with blueprint
"vehicle.lincoln.mkz_2020", with color
Color(0.3, 0.3, 0.3), with behavior
Behavior_car2
```

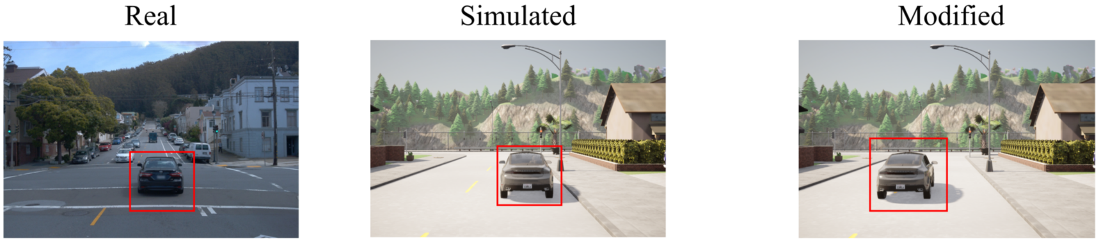
Figure 11: We can manipulate the position and rotation of the vehicle, as highlighted in the box. This indicates that we move the vehicle position closer to the ego and rotate the vehicle at 20 degrees relative to the forward-facing direction. This can be used as a test scenario where a vehicle has cut in at an angle, presenting a safety hazard.



```
car1 = new Car offset by (-2.3, 21.5), with
blueprint "vehicle.lincoln.mkz_2020", with
color Color(0.3, 0.3, 0.3), with behavior
Behavior_car1
```

```
car1 = new Car offset by (-2.3, 21.5), facing
(p[2] + math.pi / 2), with blueprint
"vehicle.lincoln.mkz_2020", with color
Color(0.3, 0.3, 0.3), with behavior
Behavior_car1
```

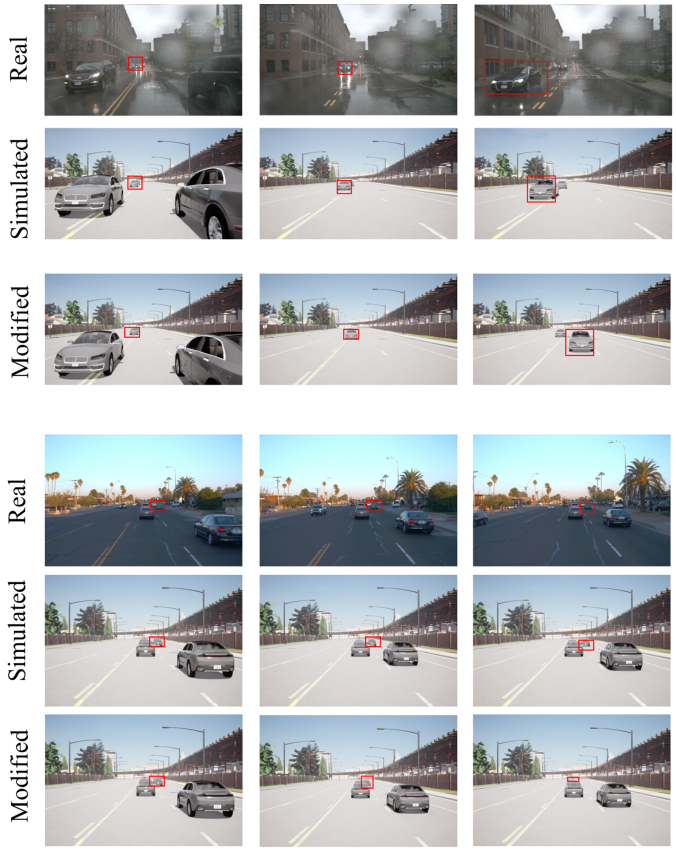
Figure 12: We show an additional example of manipulation of the position and rotation of the vehicle. This shows a situation where a vehicle has stopped on the road at a dangerous angle.



```
car3 = new Car offset by (0.3, 11.7), with
blueprint "vehicle.lincoln.mkz_2020", with
color Color(0.3,0.3,0.3), with behavior
Behavior_car3
```

```
car3 = new Car offset by (-1.5, 11.7), with
blueprint "vehicle.lincoln.mkz_2020", with
color Color(0.3,0.3,0.3), with behavior
Behavior_car3
```

Figure 13: We show an example of manipulating the vehicle position so that the vehicle is sitting at an arbitrary position on the road.



```
behavior Behavior_car7():
    do CustomFollowLaneBehavior(15.02), for 0.5 seconds
    do CustomFollowLaneBehavior(15.05) for 0.5 seconds
    do CustomFollowLaneBehavior(15.17) for 0.5 seconds
    do CustomFollowLaneBehavior(15.30) for 0.5 seconds
```

```
behavior Behavior_car7():
    do CustomFollowLaneBehavior(15.02) for 0.5 seconds
    do CustomChangeLaneBehavior(15.05,lateral_speed=-2)
        for 0.5 seconds
    do CustomChangeLaneBehavior(15.17,lateral_speed=-2)
        for 0.5 seconds
    do CustomChangeLaneBehavior(15.30,lateral_speed=-2)
        for 0.5 seconds
```

```
behavior Behavior_car3():
    do CustomFollowLaneBehavior(0.86) for 0.5 seconds
    do CustomFollowLaneBehavior(0.83) for 0.5 seconds
    do CustomFollowLaneBehavior(0.92) for 0.5 seconds
    do CustomFollowLaneBehavior(0.91) for 0.5 seconds
```

```
behavior Behavior_car3():
    do CustomFollowLaneBehavior(0.86) for 0.5 seconds
    do CustomChangeLaneBehavior(0.83, lateral_speed=2.0)
for 0.5 seconds
    do CustomChangeLaneBehavior(0.92, lateral_speed=2.0)
for 0.5 seconds
    do CustomChangeLaneBehavior(0.91, lateral_speed=2.0)
for 0.5 seconds
```

Figure 14: We show how we can manipulate the trajectory of the vehicles. We replace the instructions of a vehicle so that it performs a lane change in the simulation. The behaviors here can cause a critical situation which the autonomous vehicle must react to.