Scenario Mining and Auto-Labeling for Promptable Driving Policies

Cainan Davidson

CMU-RI-TR-26-74

July 14, 2026

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:

Professor Deva Ramanan, Chair
Professor Yonatan Bisk
Ayush Jain, *Carnegie Mellon University*

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics

© Cainan Davidson, 2026

In memory of Tristan White

Acknowledgments

Looking back, it is incredible to see how much I have grown as a researcher over the past two years. I owe this growth primarily to two people. First, I would like to thank my fantastic PhD mentor Neehar Peri. Aside from being constantly available for discussion, he was pulling strings in the background to ensure my success at every step of the way. This ranged from explicitly putting me in contact with employment opportunities to subtly letting me stumble around in my research to maximize my personal growth. I also have to thank Deva Ramanan, who is the best advisor I could have asked for. His passion for science and clear communication has been infectious. Every discussion I had with him was filled with enthusiasm. He has an incredible ability to get people to say what they want to say simply by asking questions. Deva is a true role model, and his mentorship has been invaluable.

I would also like to acknowledge the butterfly effect in being where I am today. Toward the end of my third year of undergrad, I received an untargeted email advertising an undergraduate research position. After submitting my application, Dr. Ethan Secor (who likely does not remember me) responded with a short message explaining that he had already hired a student but recommended I apply to a little-known fellowship that would be closing later the next day. That fellowship turned into my first foray into machine learning for the physical world and made me confident that I wanted to pursue further education. Had Dr. Secor not spent five minutes writing a personal response, I can confidently say that I would not be here today.

Finally, I would like to thank the following groups of people for being especially formative and supportive. I thank all the great math teachers I had in middle and high school: Juliana Hrasky, Derrick Steeve, Heather Behrendt, Thad Nelson, Alex Herrington, and Annette Hamilton. I thank all of my friends back in Iowa. I thank my undergraduate research mentors: Aishwarya Pawar, Biewen Li, and Todd Kingston. I thank the Argoverse Challenge collaborators and sponsors: Ishan Khatri, StackAV, and Uber AV Labs. I thank my mentors at Overland AI: Alec Reed and James Hays. I thank my labmates (whose work is highly influential on mine): Bart Duisterhof, Chancharik Mitra, Gautam Gare, Jacob Yeung, Jay Karhade, Jeff Tan, Khiem Vuong, Lucky Nayak, Manu Guar, Mehar Khurana, Nikhil Keetha, Qi Wu, Sally Chen, Tarasha Khurana, Yiming Gong, Zihan Wang, and Zhiqiu Lin. I thank

my peers in RI, especially: Abhishek Iyer, Ranai Srivastav, Soumya Teotia, Trinity Chung, and William Hered. Most importantly, I thank my family: Mom, Dad, Josie, and Ryder.

A significant portion of this thesis is related to the field of autonomous vehicles. The potential of this technology to save lives is close to my heart. There was a day in eighth grade where the school counselor came into the classroom to break the news that one of our classmates had died. We learned our classmate, Tristan White, was killed after being struck by a speeding car while running on a gravel road. I was not super close with Tristan, but I will never forget the sobs from two of his friends that I sat next to. Each year, there are tens of thousands more people with a story like Tristan's and millions more people with a story like his friends' and family's. It is extremely motivating for me to make this story less common.

Abstract

The largest remaining problem in autonomous vehicle (AV) development is ensuring reasonable and safe decision-making in the long-tail of edge cases that are poorly represented in training data. In this thesis, we discuss data-centric methods for building safer policies in the structured, data-rich domain of on-road driving and the unstructured, data-scarce domain of off-road driving. We first introduce RefAV, a benchmark for retrieving interesting and safety critical scenarios from uncurated driving logs. We revisit spatio-temporal scenario mining through the lens of recent vision-language models (VLMs) to detect whether a described scenario occurs in a driving log and, if so, precisely localize it in both time and space. Notably, we find that naively repurposing existing VLMs yields poor performance, suggesting that scenario mining presents unique challenges. We discuss the competition held around the RefAV benchmark and share insights from the community. The second half of this thesis explores counterfactual auto-labeling as a method to make vision-language-action (VLA) models more controllable during off-road navigation. We find that training naively on VLM-generated commands is not enough to elicit general language following, as a sample's image observation is often more predictive of the future trajectory than the language command. Instead, we augment the observation with alternative trajectories collected from other times the robot visited a particular location. We perform closed loop evaluation in 3D reconstructions of previously unseen environments. We show our dataset annotation method improves the ability of an agent to navigate autonomously to waypoints hundreds of meters away while enabling language-based interventions at important decision points.

Table of Contents

List of Figures vii

List of Tables x

1 **Introduction** 1

2 **RefAV: Towards Planning-Centric Scenario Mining** 3

2.1 Introduction 3

2.2 Related Works 6

2.3 RefAV: Scenario Mining with Natural Language Descriptions 8

2.3.1 Creating RefAV 8

2.3.2 Scenario Mining Baselines 11

2.4 Experiments 13

2.4.1 Empirical Analysis of Results 15

2.5 Conclusion 17

3 **CAL: Counterfactual Auto-Labeling for Promptable Off-Road Driving Policies** 18

3.1 Introduction 19

3.2 Related Works 20

3.3 Methods 21

3.3.1 Counterfactual Auto-labeling 21

3.4 Experiments . . . 24
3.4.1 Open-Loop Evaluation . . . 26
3.4.2 Closed-Loop Evaluation . . . 27
3.4.3 Ablations . . . 28
3.5 Conclusion . . . 30
3.5.1 Limitations . . . 30
Bibliography . . . 31
Appendix A Supplemental Material for RefAV . . . 42
A.1 Implementation Details . . . 42
A.2 Token Statistics for LLM Baselines . . . 46
A.3 Comparison of Recent Datasets . . . 47
A.4 Scenario Mining Annotation Tool . . . 51
A.5 Referential Tracking Evaluation of `Related` and `Other` Objects . . . 51
A.6 Impact of Sampling Rate on RefProg . . . 53
A.7 More Dataset Statistics . . . 53
A.8 Summary of Competition Top Performers . . . 56
A.9 Examples of RefProg's Synthesized Programs . . . 56
A.10 API Listing . . . 61
Appendix B Supplemental Material for CAL . . . 80
B.1 Language Annotation Details . . . 80
B.2 VLA Training Implementation . . . 83
B.3 Closed-Loop Simulation Implementtion . . . 84

List of Figures

Figure 2.1 **Scenario Mining Problem Setup.** Given a natural language prompt such as `vehicle making left turn through ego-vehicle's path while it is raining`, our problem setup requires models to determine whether the described scenario occurs within a 20-second driving log, and if so, precisely localize the referred object in 3D space and time from raw sensor data (LiDAR, 360° ring cameras, and HD maps). Based on the example above, a VLM should localize the start and end timestamps and 3D location of the red Mini Cooper executing a "Pittsburgh left" through the ego-vehicle's path with a 3D track. Notably, the "Pittsburgh left" is a regional driving practice where a driver quickly makes a left turn before oncoming traffic proceeds. Although common in Pittsburgh, this maneuver is technically illegal. Therefore, we argue that scenario mining is critical for validating end-to-end autonomy in order to build a comprehensive safety case. Note that `referred objects` are shown in green, `related objects` in blue, and `other objects` in red. 4

Figure 2.2 **RefAV Dataset Creation.** First, we define a set of 28 atomic functions that identify the state of an object track, its relationship with other objects (stored in an underlying scene graph), and a set of boolean logical operators to support function composition. Next, we prompt an LLM to permute these atomic functions and generate a program and corresponding natural language description. Finally, we execute the generated code on ground-truth tracks and visualize the referred object track to manually verify that the program output matches the natural language prompt. Code that generates an incorrect video is modified by an annotator and re-executed. We sample valid programs to maximize scenario diversity in our dataset. 8

Figure 2.3 **Examples of Multi-Agent Interactions.** We visualize representative examples from RefAV to highlight the diversity of our dataset. In (a), we capture the interactions between vulnerable road users and vehicles at a crowded intersection. Scenario (b) presents an atypical instance of a common multi-agent interaction (e.g. `pedestrian walking a dog`). In (c), we show a complex ego-vehicle trajectory that involves multiple moving vehicles. Scenario (d) illustrates an example of a rare multi-object interaction. In (e), we highlight a scenario that might require evasive maneuvers from the ego-vehicle (e.g. the occluded pedestrian might cross the path of the ego-vehicle). Finally, subfigure (f) visualizes a scenario with a multiple-step relationship (e.g. the official signaler is standing inside of a construction zone). Note that we show `referred objects` in green, `related objects` in blue, and all `other objects` in red. **10**

Figure 2.4 **Method Overview.** RefProg is a dual-path method that independently generates 3D perception outputs and Python-based programs for referential grounding. Given raw LiDAR and RGB inputs, RefProg runs a offline 3D perception model to generate high quality 3D tracks. In parallel, it prompts an LLM to generate code to identify the referred track. Finally, the generated code is executed to filter the output of the offline 3D perception model to produce a final set of `referred objects`, `related objects`, and `other objects`. **12**

Figure 3.1 **Counterfactual Autolabeling Pipeline.** We use precise GPS odometry to identify intersecting trajectories across robot logs. These intersections serve as natural decision-making points. **22**

Figure 3.2 **Contrastive vs Naive Annotation Example.** Overlaying multiple trajectories on an image encourages an annotator to describe details that differentiate similar trajectories, which we find desirable for more general language following. The orange box indicates the prompt given to the annotator VLM. The blue boxes contain the VLM-generated language commands, which are later used to train the VLA. **23**

Figure 3.3 **Language Instruction Examples.** We use Gemini-3-Pro to label 3,600 intersections with language commands and relevant image points. We find the majority of the annotations to be reasonable. **24**

Figure 3.4 **Imitation Learning on Tartandrive.** (a) shows the geographic splits for training and testing on Tartandrive. (b) Before data augmentation, we filter the train dataset to 187,937 training samples. We err on the side of keeping dubious trajectories, such as reversing with only a front camera, because they expose the model to a more diverse set of language instructions. **25**

Figure 3.5 **Language Interventions in Closed-Loop.** We simulate language interventions from a human operator by creating areas where an agent receives a command when it enters. By strategically intervening with language commands, we can guide the agent through the 3DGS environment toward its goal. 29

Figure A.1 **Manual Annotation Tool**. We create an annotation tool to assist with labeling manually defined scenarios. Our tool allows us to quickly annotate multi-object referential tracks in AV2. 52

Figure A.2 **Occurrences within RefAV by Scenario.** RefAV contains 407 unique scenario descriptions. We display 11 representative descriptions picked uniformly. After the diversity sampling process, RefAV includes about 40 positive matches for common scenarios such as `vehicle making a right turn`. Less common scenarios such as `moving bus being crossed by a bicycle` occur only in only a handful of logs across the ArgoVerse 2 dataset. RefAV includes negative examples for almost all procedurally defined scenarios. Manually annotated scenarios such as `another autonomous vehicle` do not have corresponding negative examples. 54

Figure A.3 **RefAV Expression Statistics.** We compute the number of referring expressions categorized by expression type, count the number of the expressions that are positive or negative, and whether they are procedurally generated or manually constructed. We also highlight the number of positive expressions that include `related object` annotations and explicit references to the ego vehicle. The bar chart on the right shows the number of objects refered to by each expression. Most expressions in RefAV refer to zero (e.g. a negative match) or one object. On average, each expression refers to 3.92 objects. 55

Figure A.4 **RefAV Object Distribution.** RefAV includes `referred objects` in all directions up to 150m away from the ego vehicle. RefAV places a special focus on the ego vehicle and objects that interact with the ego vehicle. Therefore, `referred objects` are disproportionately located on the road in front of the ego vehicle. Both `referred object` heatmaps are in the ego vehicle coordinate frame. 55

List of Tables

Table 2.1 **Comparison to Other Benchmarks.** Language-based 3D scene understanding has been extensively studied in the context of referential multi-object tracking (RMOT) and multi-modal visual question answering (VQA). Different from prior work, we address the problem of spatio-temporal scenario mining. Specifically, RefAV is based on Argoverse 2, which provides 3D track-level annotations for 30 categories at 10 Hz. Although RefAV does not include as many referential expressions as prior work (e.g. OmniDrive [1] and nuGrounding [2]), our referential annotations focus on capturing diverse multi-agent interactions. Lastly, RefAV includes negative prompts, which allows us to more accurately measure scenario mining performance. 11

Table 2.2 **Experimental Results.** We evaluate several zero-shot referential tracking baselines. We find that RefProg significantly outperforms all other zero-shot baselines. Notably, filtering by referred class is a particularly strong baseline, outperforming image-embedding similarity. Interestingly, directly using LLM APIs as a black box outperforms ReferGPT's hand-crafted approach. All winning submissions to our challenge build upon RefProg. 14

Table 2.3 **Impact of Different LLMs on Code Synthesis Quality.** We evaluate the impact of using different LLMs for program synthesis in the RefProg pipeline. Interestingly, we find that Claude 3.5 Sonnet has the lowest program failure rate (e.g. 99.5% of Claude 3.5 Sonnet's generated programs were valid, compared to 81.9% for Qwen 2.5B Instruct), while Claude 3.7 Sonnet achieves the highest HOTA-Temporal. 16

Table 2.4 **Zero-Shot Evaluation on nuPrompt.** We evaluate RefProg on nuPrompt and achieve state-of-the-art accuracy, highlighting the strong generalization of our approach. Importantly, we do not modify RefProg's atomic action definitions. 17

Table 3.1 **Validating our Closed-Loop Environments.** We evaluate the real2sim gap using standard reconstruction metrics. Since control is abstracted away, we quantify the sim2real gap using open-loop planning metrics. . . . 26

Table 3.2 **Effect of Language Annotation Method on Instruction Following.** We find that naive language annotations do not provide enough signal for counterfactual instruction following to emerge (i.e. the model learns that it does not need to pay attention to the language). 26

Table 3.3 **Open-Loop Performance by Base VLM**. Zero-shot rows correspond to models without CAL finetuning, while CAL rows use additional finetuning on the seen terrain. We find that all models fail in the zero-shot setting and that the base VLM affects trajectory quality. 27

Table 3.4 **Closed-Loop Performance in Different Environments** We evaluate the ability of the model to adhere to an trajectory from an expert-driven log. In the forest environment, it takes three language interventions to keep the model from diverging from the expert trajectory. Deviation indicates average distance from the logged expert trajectory in meters. . . 28

Table 3.5 **Ablation on CAL Annotations**. Training on the counterfactual trajectories and ensuring annotation coverage near decision points are both important for CAL. 29

Table A.1 **Token Efficiency by LLM Baseline** RefProg achieves the highest scenario mining performance for the lowest cost. Cost is in USD for the OpenAI Platform in November 2025. We report the total number of tokens used millions. CIS refers to code interpreter sessions. 47

Table A.2 **Referential Tracking Accuracy for `Related` and `Other` Objects.** We present RefProg's referential tracking accuracy for `related objects` and `other objects`. We find that HOTA (for the base tracker) is similar to HOTA-Temporal (`Other`) because most objects are not relevant to the referential prompt. 53

Table A.3 **Impact of Sampling Rate.** We evaluate RefProg's performance with subsampled inputs to measure the relative impact on referential tracking accuracy. We find that standard tracking performance drops by 5% when subsampling due to the difficulty of associating sparse detections. 53

Table B.1 **Training Hyperparameters.** 83

Chapter 1

Introduction

A grand challenge in robotics is ensuring reasonable and safe decision-making across the full range of situations a robot may encounter, including the long-tail of edge cases that are poorly represented in training data. Nowhere is this more apparent than in the domain of autonomous driving. In 1995, researchers at Carnegie Mellon University were able to drive across the United States in a car that was autonomously steering 98% of the time [3]. Since then, there has been over 100 billion dollars in external investment in the AV industry [4] to reliably solve the remaining 2% of driving.

Today, several companies are rapidly scaling up autonomous vehicle deployments on public roads. Waymo, for example, is completing $500,000$ passenger rides per week as of March 2026 [5]. This success can largely be attributed to advances in machine learning and the relative ease of collecting the data these algorithms use. The environments are purpose-built for vehicles and teleoperation data can be collected by people doing the driving they already would be otherwise. During fleet testing and deployment, each vehicle collects terabytes of multi-modal observational data. It is a major challenge to sort through the firehose of incoming data to identify specific edge cases needed for active learning and building a comprehensive safety-case.

Another emerging use case for autonomous vehicles is in off-road driving (e.g. for forestry, agriculture, or mining). In this domain, the environments are less structured, data is not as readily available, and evaluation infrastructure is not as mature. This naturally calls for a more general set of priors and intelligent methods to leverage the data that is available. In this thesis, we discuss data-centric methods for building safer policies in the structured, data-rich domain of on-road driving and the unstructured, data-scarce domain of off-road driving.

The first half of this thesis discusses RefAV, a benchmark for retrieving interesting and safety critical scenarios from uncurated on-road driving datasets. Traditional scenario mining

techniques are error-prone and prohibitively time-consuming, often relying on hand-crafted structured queries. In this work, we revisit spatio-temporal scenario mining through the lens of recent vision-language models (VLMs) to detect whether a described scenario occurs in a driving log and, if so, precisely localize it in both time and space. To address this problem, we introduce RefAV, a large-scale dataset of 10,000 diverse natural language queries that describe complex multi-agent interactions relevant to motion planning derived from 1000 driving logs in the Argoverse 2 Sensor dataset. We evaluate several referential multi-object trackers and present an empirical analysis of our baselines. Notably, we find that naively repurposing off-the-shelf VLMs yields poor performance, suggesting that scenario mining presents unique challenges. Lastly, we discuss our recently held competition and share insights from the community. Our code and dataset are available on **Github** and **Argoverse**.

The second half of this thesis explores counterfactual auto-labeling as a method to make vision-language-action (VLA) models more controllable during off-road navigation. In on-road driving, high-level decision making typically exploits a map that directly provides turn and lane change commands to a low-level policy controller. We focus on unstructured off-road environments where such maps do not exist, making the decision-making hierarchy unclear. We therefore aim to train a policy that is responsive to both long-horizon waypoints and free-form language commands such as “stay on the gravel" or “avoid the puddle". We find that training naively on VLM-generated commands is not enough to elicit language following, as the image observation is often more predictive of the future trajectory than the language command due to natural biases in navigation datasets. Instead, we augment the observation with alternative trajectories collected from other times the robot visited a particular location. We show that adding these "counterfactual" trajectories (and their associated commands) decreases language following error by 0.28m ADE over a naively auto-labeled dataset. Finally, we perform closed loop evaluation in 3D reconstructions of previously unseen environments. We show our dataset annotation method improves the ability of an agent to navigate autonomously to waypoints hundreds of meters away while enabling language-based interventions at important decision points.

Chapter 2

RefAV: Towards Planning-Centric Scenario Mining

Abstract: Autonomous Vehicles (AVs) collect and pseudo-label terabytes of multi-modal data localized to HD maps during normal fleet testing. However, identifying interesting and safety-critical scenarios from uncurated driving logs remains a significant challenge. Traditional scenario mining techniques are error-prone and prohibitively time-consuming, often relying on hand-crafted structured queries. In this work, we revisit spatio-temporal scenario mining through the lens of recent vision-language models (VLMs) to detect whether a described scenario occurs in a driving log and, if so, precisely localize it in both time and space. To address this problem, we introduce RefAV, a large-scale dataset of 10,000 diverse natural language queries that describe complex multi-agent interactions relevant to motion planning derived from 1000 driving logs in the Argoverse 2 Sensor dataset. We evaluate several referential multi-object trackers and present an empirical analysis of our baselines. Notably, we find that naively repurposing off-the-shelf VLMs yields poor performance, suggesting that scenario mining presents unique challenges. Lastly, we discuss our recently held competition and share insights from the community. Our code and dataset are available on **Github** and **Argoverse**.

2.1 Introduction

Autonomous Vehicle (AV) deployment on public roads has increased significantly in recent years, with Waymo completing more than 500,000 rides per week as of March 2027 [5]. Despite the maturity of such autonomous ride-hailing services, AVs can still sometimes require manual interventions [6, 7] and can be prone to accidents [8]. Although data-driven simulators are an integral component in establishing a safety case [9–12], validating end-to-end autonomy with real-world operational data remains a critical part of the testing stack. However, identifying interesting and safety-critical scenarios from uncurated real-world driving logs is akin to

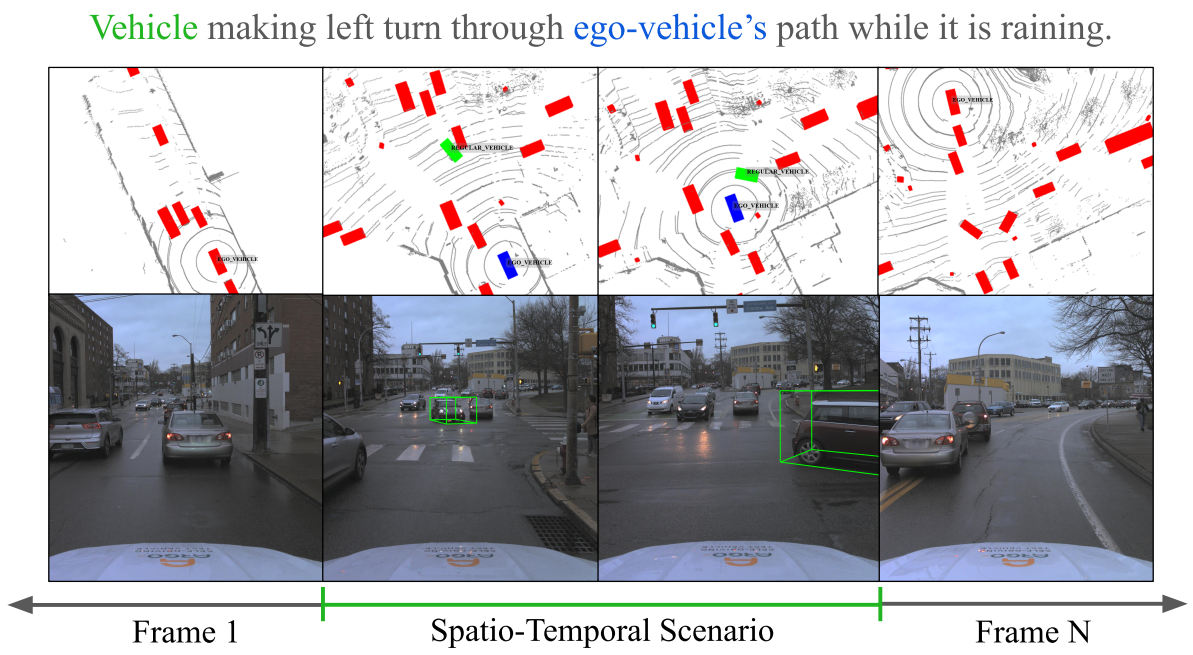


Figure 2.1. **Scenario Mining Problem Setup.** Given a natural language prompt such as `vehicle making left turn through ego-vehicle's path while it is raining`, our problem setup requires models to determine whether the described scenario occurs within a 20-second driving log, and if so, precisely localize the referred object in 3D space and time from raw sensor data (LiDAR, 360° ring cameras, and HD maps). Based on the example above, a VLM should localize the start and end timestamps and 3D location of the red Mini Cooper executing a "Pittsburgh left" through the ego-vehicle's path with a 3D track. Notably, the "Pittsburgh left" is a regional driving practice where a driver quickly makes a left turn before oncoming traffic proceeds. Although common in Pittsburgh, this maneuver is technically illegal. Therefore, we argue that scenario mining is critical for validating end-to-end autonomy in order to build a comprehensive safety case. Note that `referred objects` are shown in green, `related objects` in blue, and `other objects` in red.

finding a "needle in a haystack" due to the scale of data collected during normal fleet operations. In this paper, we revisit the task of spatio-temporal scenario mining with recent vision-language models (VLMs) to identify interesting multi-agent interactions using natural language (cf. Fig. Figure 2.1).

Status Quo. Although language-based 3D scene understanding has been extensively studied in the context of referential multi-object tracking (RMOT) [2, 13, 14], multi-modal visual question answering (VQA) [15–17], and VLM-based motion planning [1, 18–20], we argue that spatiotemporal scenario mining presents unique challenges. RMOT extends referential grounding by associating referred objects over time. However, unlike RMOT, scenario mining does not guarantee that referred objects exist in a given driving log. Next, multi-modal VQA extends VQA with additional visual modalities like LiDAR. Although spatio-temporal scenario mining also leverages multi-modal input (e.g. RGB, LiDAR and HD maps), we require that methods output 3D tracks instead of text-based answers. Lastly, VLM-based motion planners directly estimate future ego-vehicle waypoints based on high-level language instructions and past sensor measurements. In contrast, scenario mining methods can reason over the full driving log to identify interactions between non-ego vehicles. Concretely, spatio-temporal scenario mining requires identifying if a described scenario occurs in a driving log from raw sensor measurements (e.g. RGB and LiDAR), and if so, precisely localizing *all* referred objects in both time and space with 3D tracks. To support this task, we propose RefAV, a large-scale dataset of $10,000$ diverse natural language queries designed to evaluate a model's ability to find a visual "needle in a haystack".

RefAV Scenario Mining Dataset. We repurpose the Argoverse 2 (AV2) Sensor dataset [21], which contains $1,000$ driving logs with synchronized LiDAR, 360° ring cameras, HD maps, and 3D track annotations for 30 categories. We curate a set of $10,000$ natural language prompts that describe interesting (and often rare) scenarios (cf. Fig. Figure 2.2 and Fig. Figure 2.3) using a combination of manual annotations and procedural generation using large language models (LLMs). Notably, AV2 annotates ground truth tracks at 10 Hz, which allows for fine-grained motion understanding. In contrast, prior referential multi-object tracking benchmarks (the established task most similar to spatio-temporal scenario mining) use nuScenes [22], which annotates ground truth tracks at 2 Hz (cf. Table 2.1). Argoverse's higher temporal resolution uniquely allows RefAV to evaluate dynamic multi-agent interactions (unlike prior work that primarily evaluates referential expressions based on static attributes like vehicle color or relative heading). Interestingly, we find that simply repurposing VLMs for scenario mining yields poor performance.

Referential Tracking by Program Synthesis. Although prior referential trackers and VLMs achieve reasonable performance with simple referential prompts (e.g. `find all cars`),

we find that such methods struggle with compositional reasoning and motion understanding (e.g. `find all cars accelerating while changing lanes`). To address this problem, we propose Referential Tracking by Program Synthesis (RefProg), a modular approach that combines off-the-shelf 3D tracks with LLMs. Inspired by recent work in program synthesis [23–25], our method uses an LLM to break down complex referential expressions into simpler compositional actions. Specifically, we define an API to describe hand-crafted atomic functions (e.g., `turning`, `accelerating`, `changing lanes`) and use an LLM to synthesize a program that composes these atomic actions corresponding to the natural language prompt. We then execute the generated program to filter off-the-shelf tracks to identify the subset of tracks that best match the described scenario (cf. Fig. **Figure 2.4**).

Contributions. We present three major contributions. First, we introduce RefAV, a large-scale dataset designed to evaluate VLMs on 3D scene understanding and spatio-temporal localization. Our extensive experiments highlight the limitations of current methods, and demonstrate the effectiveness of our proposed program synthesis-based approach. Lastly, we highlight the results of our recent **CVPR 2025 challenge** hosted in conjunction with the **Workshop on Autonomous Driving**.

2.2 Related Works

Referential Grounding and Tracking are long-standing challenges in vision-language understanding. Early datasets like RefCOCO, RefCOCO+, and RefCOCOg [26–28] helped popularize the task, which was later adapted for autonomous driving [29, 30]. Although prior work focused on single-frame visual grounding, ReferKITTI [13] formalizes the problem of 2D referential multi-object tracking (RMOT), which extends referential object detection by associating referred objects over time. ReferKITTI-v2 [31] expands ReferKITTI's manual annotations using large language models (LLMs) to improve prompt diversity. Further, LaMOT [32] consolidates 1,660 sequences from four tracking datasets to create a large-scale unified RMOT benchmark.

Referential tracking methods can be broadly classified into two-stage (e.g. separately track and filter predictions based on natural language cues) and end-to-end approaches. iKUN [33] proposes a plug-and-play knowledge unification module to facilitate language grounding for off-the-shelf trackers. Similarly, ReferGPT [34] presents a zero-shot method for filtering off-the-shelf 3D tracks using CLIP [35] scores and fuzzy matching. In contrast, JointNLT [36] proposes a unified visual grounding and tracking framework, and MMTrack [37] reformulates tracking as token generation. OVLM [38] introduces a memory-aware model for RMOT, while ReferFormer [39] leverages language-conditioned object queries. More recently, nuPrompt

[14] extends 2D referential tracking to 3D with RGB input. Different from nuPrompt, RefAV addresses 3D *multi-modal* referential tracking and dynamic scene understanding.

Visual Question Answering with VLMs has improved significantly in recent years due to large-scale multi-modal pre-training. Vision-language models (VLMs) like LLaVA [40, 41], BLIP-2 [42], and Qwen2.5-VL [43] show strong generalization across diverse domains [15, 17, 44]. However, state-of-the-art models still struggle with spatial reasoning and grounding [45, 46]. To address this issue, SpatialRGPT [47] and SpatialVLM [48] generate large-scale monocular depth pseudo-labels to improve 3D reasoning. Despite the effectiveness of such methods, zero-shot prompting yields poor compositional reasoning performance. Instead, VisProg [24] and ViperGPT [23] use LLMs to generate executable code and use tools like OwlViT [49], CLIP [35], MiDaS [50], and GLIP [51] for spatial understanding. However, existing program synthesis approaches primarily focus on single-frame reasoning. In contrast, we are interested in understanding dynamic multi-agent interactions from video sequences. We take inspiration from existing methods and apply similar program synthesis-based approaches for spatio-temporal scenario mining.

Vision-Language Models for 3D Understanding have been extensively explored in the context of representation learning, open-vocabulary 3D perception, and end-to-end driving. SLidR [52] distills 2D features into 3D point clouds for cross-modal learning. SEAL [53] incorporates SAM [54] to produce class-agnostic 3D segments, while SA3D [55] leverages SAM and NeRFs for object segmentation. Similarly, recent work like Anything-3D [56] and 3D-Box-Segment-Anything integrate VLMs and 3D detectors (e.g., VoxelNeXt [57]) for interactive reconstruction and labeling. VLMs have been used extensively in open-vocabulary perception for autonomous driving. UP-VL [58] distills CLIP features into LiDAR data to generate amodal cuboids. Recent work uses 2D VLMs to generate 3D pseudo-labels for open-vocabulary perception, enabling zero-shot LiDAR panoptic segmentation [59, 60] and 3D object detection [61–63]. While traditional grounding methods [14] struggle with complex instructions, multi-modal LLMs [19, 20, 64] demonstrate impressive visual understanding. However, such methods [1, 65, 66] typically produce scene-level analysis for end-to-end driving rather than precise instance localization. In contrast, our framework combines language grounding with precise geometric localization from offline 3D perception methods for more accurate vision-language reasoning.

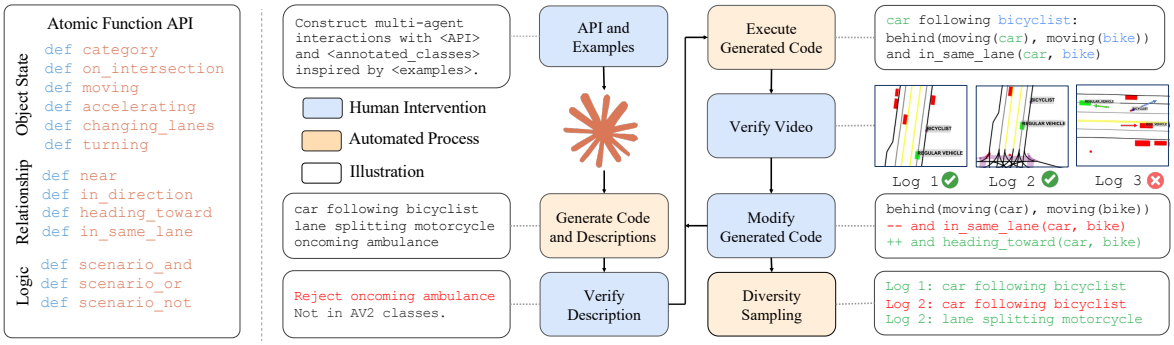


Figure 2.2. **RefAV Dataset Creation.** First, we define a set of 28 atomic functions that identify the state of an object track, its relationship with other objects (stored in an underlying scene graph), and a set of boolean logical operators to support function composition. Next, we prompt an LLM to permute these atomic functions and generate a program and corresponding natural language description. Finally, we execute the generated code on ground-truth tracks and visualize the referred object track to manually verify that the program output matches the natural language prompt. Code that generates an incorrect video is modified by an annotator and re-executed. We sample valid programs to maximize scenario diversity in our dataset.

2.3 RefAV: Scenario Mining with Natural Language Descriptions

In this section, we present our approach for curating 10,000 natural language prompts (Sec 2.3.1) and describe five zero-shot scenario mining baselines (Sec 2.3.2).

2.3.1 Creating RefAV

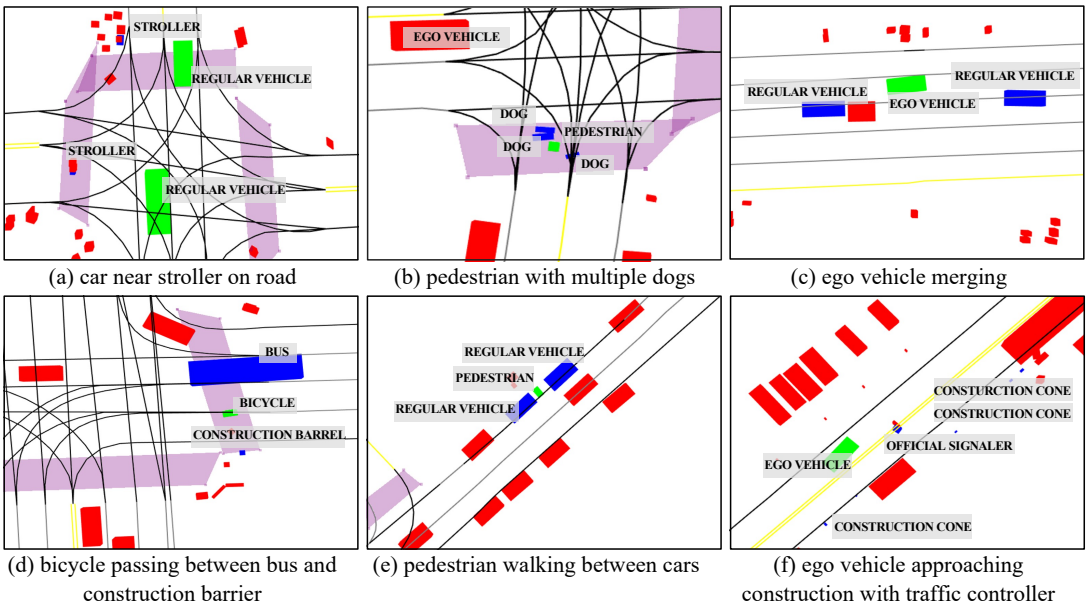
Unlike prior benchmarks that primarily evaluate referential expressions based on static attributes like vehicle color or relative heading (e.g. `find the red car to the left`), we focus on mining interesting and safety-critical scenarios relevant for motion planning (cf. Fig. Figure 2.1). We take inspiration from recent planning benchmarks like nuPlan [9] to identify such planning-centric scenarios. In particular, nuPlan introduces a set of 80 scenarios considered relevant for safe motion planning. We use this as a template to identify similar scenarios within AV2.

Why Use the Argoverse 2 (AV2) Sensor Dataset? The AV2 sensor dataset contains 1000 15 – 20 second logs with synchronized sensor measurements from seven ring cameras and two LiDAR sensors. Moreover, the dataset includes HD maps with lane markings, crosswalk polygons, and various lane types (e.g. vehicle, bus, and bike lanes). Notably, AV2 annotates 30 object categories with track-level annotations at 10 Hz. We choose to build RefAV on

top of AV2 because prior datasets like nuScenes [22] only annotate ground-truth tracks at a lower temporal resolution (e.g. 2Hz vs. 10Hz), making it more difficult to extract fine-grained motion. We ablate this in the Appendix A.6. In addition, KITTI [67] and Waymo Open Dataset [68] only label a limited number of categories (e.g. `car`, `pedestrian`, `bicycle`) and do not have HD maps, making it difficult to evaluate diverse multi-agent interactions. Although AV2 annotates objects up to 150m away from the ego vehicle, we clip all object tracks at 50m. We find that current 3D perception models struggle with long-range detection and tracking [69], suggesting that the community is not ready to address spatio-temporal scenario mining at range.

LLM-based Procedural Scenario Generation. Our key insight is that many complex scenarios (e.g. `find all cars accelerating while changing lanes`) can be broken down into simpler atomic actions (e.g. `find cars`, `accelerating`, and `changing lanes`), and a large set of such atomic actions can be composed to generate new diverse scenarios. To this end, we first define 28 atomic functions based on nuPlan's list of planning scenarios (cf. Figure 2.2 left). We include a full API listing in the Appendix A.10. To generate a new scenario definition, we provide an LLM (in our case, Claude 3.7 Sonnet) with the full API listing, along with in-context examples of real compositions. We prompt the LLM to generate permutations of the atomic functions and describe the generated code with a natural language description (cf. Figure 2.2 right). We execute the generated code to filter ground-truth tracks from all 1000 logs to identify true positive matches. We aggregate all true positive log-prompt matches (>50K) and sample 8,000 true positive log-prompt pairs that maximize scenario diversity. Lastly, we randomly sample 2,000 true negative log-prompt pairs. Our automatic scenario generation pipeline has an average success rate of 70%.

Verifying Procedurally Generated Scenarios. Although LLM-based procedural generation allows us to generate diverse scenarios at scale, this process is not perfect and requires extensive manual validation (cf. Figure 2.2 right). First, we verify that the generated descriptions match the generated code. We then manually review video clips of true positive log-prompt matches to ensure that the generated natural language description accurately describes the localized tracks. Notably, we find two common error modes. First, the LLM often defines the prototypical case of a scenario, but misses or incorrectly includes edge cases due to under or overspecification. For example, given a description `car following a bicyclist`, the generated code identifies instances where a moving car is behind a moving bike while traveling in the same road lane. This definition includes the false positive edge case where the car and bicyclist are traveling in opposite directions. Second, we find that LLMs often reverse the relationship of `referred` and `related` objects. For example, given the description `bicycles in front of the car`, the LLM generates code that corresponds



(a) car near stroller on road (b) pedestrian with multiple dogs (c) ego vehicle merging

(d) bicycle passing between bus and construction barrier (e) pedestrian walking between cars (f) ego vehicle approaching construction with traffic controller

Figure 2.3. **Examples of Multi-Agent Interactions.** We visualize representative examples from RefAV to highlight the diversity of our dataset. In (a), we capture the interactions between vulnerable road users and vehicles at a crowded intersection. Scenario (b) presents an atypical instance of a common multi-agent interaction (e.g. `pedestrian walking a dog`). In (c), we show a complex ego-vehicle trajectory that involves multiple moving vehicles. Scenario (d) illustrates an example of a rare multi-object interaction. In (e), we highlight a scenario that might require evasive maneuvers from the ego-vehicle (e.g. the occluded pedestrian might cross the path of the ego-vehicle). Finally, subfigure (f) visualizes a scenario with a multiple-step relationship (e.g. the official signaler is standing inside of a construction zone). Note that we show `referred objects` in green, `related objects` in blue, and all `other objects` in red.

to `car behind the bicycles`. Based on the videos, human annotators make the necessary changes to the scenario definition, and the modified code is re-executed across all logs. It takes about 3 minutes to verify and edit each scenario definition. In total, we spent 200 hours verifying scenario definitions.

Manual Scenario Annotation. Despite the versatility of our procedural generation approach, many interesting interactions cannot be easily identified by composing atomic functions. Therefore, we manually inspect all videos in the validation and test sets to identify interesting driving behaviors (cf. **Figure 2.3**). Further, we manually annotate weather (e.g. `clear`, `cloudy`, `rain`, `snow`, and `fog`) and lighting conditions (e.g. `daylight`, `dusk/dawn`, and `night`) in the validation and test sets as these attributes are relevant for establishing a safety case [70] in all driving conditions. See the Appendix **A.4** for more details about our annotation workflow.

Dataset Statistics. As shown in Table 2.1, our dataset uniquely addresses the task of spatio-temporal scenario mining, whereas other datasets focus on object detection, referring multi-object tracking (RMOT), or visual question answering (VQA). Further, RefAV is uniquely built on Argoverse 2 [21], whereas most existing datasets rely on nuScenes [22] or KITTI [67]. Next, RefAV provides track-level annotations at a higher frequency of 10Hz, compared to the typical 2Hz in other datasets, allowing for more fine-grained temporal analysis. While datasets like nuGrounding [2] and nuScenes-QA [15] include large numbers of expressions, RefAV emphasizes a diversity of annotation types, including referring expressions that capture dynamic multi-agent interactions, weather, and lighting conditions. Lastly, RefAV is one of only two datasets to support negative prompts.

Table 2.1. **Comparison to Other Benchmarks.** Language-based 3D scene understanding has been extensively studied in the context of referential multi-object tracking (RMOT) and multi-modal visual question answering (VQA). Different from prior work, we address the problem of spatio-temporal scenario mining. Specifically, RefAV is based on Argoverse 2, which provides 3D track-level annotations for 30 categories at 10 Hz. Although RefAV does not include as many referential expressions as prior work (e.g. OmniDrive [1] and nuGrounding [2]), our referential annotations focus on capturing diverse multi-agent interactions. Lastly, RefAV includes negative prompts, which allows us to more accurately measure scenario mining performance.

Dataset	Base Data	Task	View	Anno. Freq.	# Expressions	# Frames	Neg. Prompts	Human Anno.
Talk2Car [29]	nuScenes	Detection	Front	Per Video	12k	9k	✗	Referring Expression
nuScenes-QA [15]	nuScenes	VQA	360°	2Hz	460k	34k	✓	None
DriveLM [18]	nuScenes	VQA	360°	2Hz	443k	5k	✗	QA Pairs
OmniDrive [1]	nuScenes	VQA	360°	2Hz	200k	34k	✗	None
Refer-KITTI [13]	KITTI	RMOT	Front	2Hz	818	7k	✗	Referring Expression
nuGrounding [2]	nuScenes	RMOT	360°	2Hz	2.2M	34k	✗	Object Attribute
nuPrompt [14]	nuScenes	RMOT	360°	2Hz	407k	34k	✗	Object Color
RefAV	Argoverse 2	RMOT & Scenario Mining	360°	10Hz	10k	155k	✓	Referring Expression, Weather, Lighting

2.3.2 Scenario Mining Baselines

We present five referential tracking baselines for scenario mining, including filtering by referred class, ReferGPT [34], image embedding similarity, LLM APIs as a black box, and referential tracking by program synthesis. We repurpose 3D trackers from the **AV2 End-to-End Forecasting Challenge**

Filtering by Referred Class. We propose a simple "blind" baseline that does not explicitly reason about multi-agent interactions. For a given natural language prompt, we use an LLM to parse the `referred object` class and only keep 3D tracks from this class. For example, for the natural language query `find all cars turning left`, we remove all predicted tracks except `cars`.

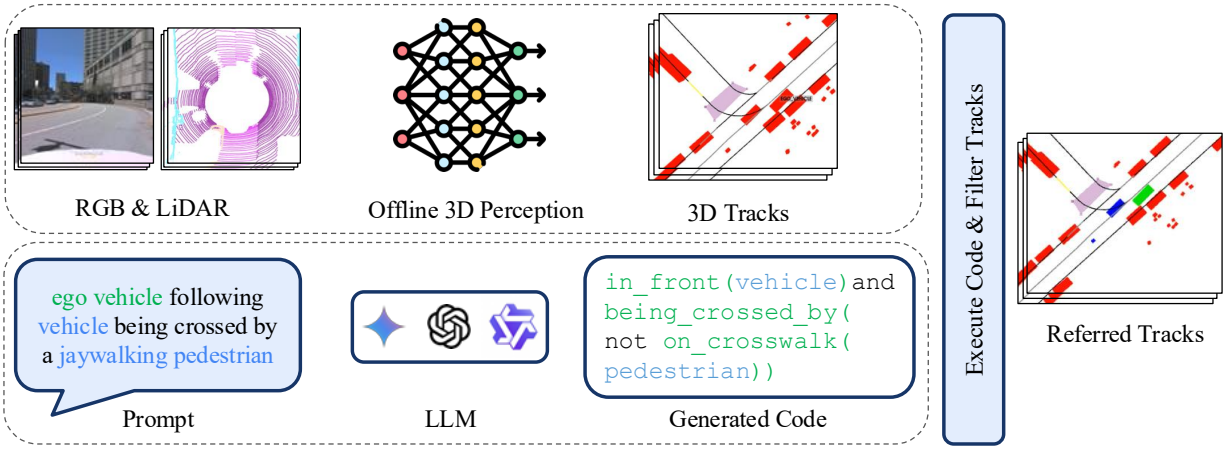


Figure 2.4. **Method Overview.** RefProg is a dual-path method that independently generates 3D perception outputs and Python-based programs for referential grounding. Given raw LiDAR and RGB inputs, RefProg runs a offline 3D perception model to generate high quality 3D tracks. In parallel, it prompts an LLM to generate code to identify the referred track. Finally, the generated code is executed to filter the output of the offline 3D perception model to produce a final set of `referred objects`, `related objects`, and `other objects`.

ReferGPT. ReferGPT [34] classifies the output of off-the-shelf 3D trackers into `referred objects` and `other objects` using CLIP text embeddings. First, ReferGPT projects the predicted 3D bounding box of each track onto the 2D image plane at each timestamp to get an image crop of each object. Next, ReferGPT parses the object's coordinates, velocity, yaw, yaw rate, and distance from the ego-vehicle. Using both the object image crop and bounding box descriptor, ReferGPT uses a VLM to generate a descriptive caption. Finally, ReferGPT uses the text cosine-similarity between the generated caption and referential prompt to score the relevance of the object track. We use GPT-5-mini as our captioning VLM and SigLIP2 [71] to compute text embeddings. We make several modifications to ReferGPT to adapt it for the spatio-temporal scenario mining problem. Since an object may be visible in multiple cameras at a single timestamp, we only caption the 2D bounding box with the largest area at each timestamp. ReferGPT selects `referred objects` by clustering tracks with the highest similarity scores. Since RefAV contains also many negative prompts, we consider a track to be a `referred object` if the similarity score between the caption text embeddings and prompt text embedding are above a fixed threshold for more than 50% of the predicted track length.

Image-Embedding Similarity. We take inspiration from ReferGPT to classify the output of off-the-shelf 3D trackers into `referred objects` and `other objects` by directly using CLIP [35] image features. We extract the track image crops in the same way as ReferGPT and compute CLIP image features for each track at each timestep. Next, we compute the cosine similarity between the per-timestep CLIP image embeddings and the CLIP text embedding of the referential prompt. We consider a track to be a `referred object` if the similarity score between the image and text embeddings are above a fixed threshold for more than 50% of the track length.

LLM API as a Black Box. We evaluate GPT-5 as a black-box scenario mining algorithm. For each log in AV2, we ask the OpenAI Responses API to identify which predictions match the referential prompt, and provide map data, city to ego-vehicle transforms, and 3D track information from an off-the-shelf tracker. GPT-5 responds with a CSV listing all `referred objects` from the initial set of predictions. Interestingly, analyzing the thinking traces of GPT-5 shows that it writes code from scratch to parse the input files and reason about multi-agent interactions.

Referential Tracking by Program Synthesis (RefProg). RefProg takes a similar approach as LLM API as a Black Box, but provides more code scaffolding to improve accuracy. Specifically, we prompt an LLM to generate code using an API listing of atomic actions for a given referential query. We execute the generated code on off-the-shelf 3D tracks to identify `referred objects`, `related objects` and `other objects` (cf. **Figure 2.4**). In addition to using API listings for estimating an object's state and relationships from 3D tracks, RefProg is given access to visual tools such as SigLIPv2 to identify visual attributes of tracked objects. Despite the similarity of RefProg and our LLM-based procedural scenario generator, we highlight two key differences. First, RefProg generates code based on the referential query, while our LLM-based procedural scenario generator synthesizes the referential query based on the generated code. We posit that the role of the scenario generator is considerably easier since there are many valid natural language prompts to describe a generated program, but significantly fewer valid programs corresponding to each natural language prompt. Lastly, code from our scenario generator is extensively modified by human annotators for correctness, while code from RefProg is executed without verification. Notably, we also evaluate RefProg on nuPrompt to demonstrate that the atomic functions generalize beyond RefAV.

2.4 Experiments

In this section, we briefly describe our evaluation metrics, provide an empirical analysis of our baselines, ablate the impact of different LLMs on RefProg's spatio-temporal scenario mining

Table 2.2. **Experimental Results.** We evaluate several zero-shot referential tracking baselines. We find that RefProg significantly outperforms all other zero-shot baselines. Notably, filtering by referred class is a particularly strong baseline, outperforming image-embedding similarity. Interestingly, directly using LLM APIs as a black box outperforms ReferGPT's hand-crafted approach. All winning submissions to our challenge build upon RefProg.

Method	HOTA ↑	HOTA-Temporal ↑	HOTA-Track ↑	Log Bal. Acc. ↑	Timestamp Bal. Acc. ↑
Ground Truth	100.0	13.3	20.5	50.0	50.0
Filter by Referred Class					
Ground Truth	100.0	21.4	33.9	52.9	57.2
LE3DE2E [72]	74.4	19.2	30.0	53.4	55.0
Image-Embedding Similarity					
Ground Truth	100.0	24.6	28.3	54.7	57.0
Le3DE2E [72]	74.4	17.2	24.4	51.1	51.1
ReVoxelDet [73]	63.1	17.1	21.2	52.3	54.4
TransFusion [74]	63.6	16.0	18.2	52.8	55.3
ReferGPT [34]					
Le3DE2E [72]	74.4	20.2	30.8	57.0	57.1
LLM APIs as a Black Box					
Le3DE2E [72]	74.4	37.2	39.2	58.4	62.3
Referential Tracking by Program Synthesis					
Ground Truth	100.0	64.8	68.7	81.1	80.7
Le3DE2E [72]	74.4	50.1	51.1	**71.8**	74.6
ReVoxelDet [73]	63.1	43.9	45.1	70.7	69.8
TransFusion [74]	63.6	48.1	46.6	70.7	68.7
Valeo4Cast [75]	69.2	41.7	44.1	62.7	69.1
Challenge Submissions					
Zeekr UMCV	74.4	**53.4**	51.0	66.3	76.6
Mi3 UCM	74.4	52.4	**51.5**	65.8	**77.5**
ZXH	74.4	52.1	50.2	66.5	76.1

accuracy, and evaluate RefProg on nuPrompt to demonstrate the robustness of our approach.

Metrics. We evaluate scenario mining methods using HOTA-Temporal and HOTA-Track (variants of HOTA [76]) to measure referential tracking performance and balanced accuracy to measure information retrieval accuracy. HOTA is a unified metric that explicitly balances detection accuracy, association, and localization, making it better aligned with human visual evaluations of tracking performance. HOTA-Temporal extends the standard HOTA metric by only considering the referred timestamps within a track as true positives. HOTA-Track is similar to HOTA-Temporal, but it does not penalize methods for incorrectly predicting the start and end of a referred action. For example, given the prompt `car turning right`, HOTA-Temporal only considers the timestamps where the car is turning right in the full referred track as true positives, whereas HOTA-Track considers all timestamps in the referred track as true positives. We also include HOTA results to contextualize standard tracking performance with the referential tracking performance of all benchmarked trackers. Additionally, we use timestamp balanced accuracy as a timestamp-level classification metric and log balanced accuracy as a log-level classification metric to evaluate how well methods can identify which

timestamps and logs contain objects that match the referential prompt, respectively. Balanced accuracy is used over binary classification metrics such as F1 due to the imbalance of positive and negative prompts. For each prompt, we categorize all objects into three groups: `referred objects`, which are the primary objects specified by the prompt; `related objects`, which are objects that interact with the referred object; and `other objects`, which are neither referred to nor relevant to the prompt. We compute both HOTA-Temporal and HOTA-Track metrics exclusively for the `referred object` class in the main paper. We report the performance on `related objects` and `other objects` in the Appendix A.5.

2.4.1 Empirical Analysis of Results

Table 2.2 presents several simple baselines for addressing spatio-temporal scenario mining. We evaluate each of the locally run zero-shot baselines using oracle ground-truth track annotations to understand how tracking accuracy and language grounding independently influence final performance. We do not evaluate *ReferGPT* and *LLM APIs as a Black Box* with the ground-truth track annotations to avoid dataset leakage. Evaluating every track as `referred object` without considering language prompts (first row) yields a HOTA-Temporal of 13.3% and HOTA-Track of 20.5%. Predicting the nearest object class that corresponds to the prompt with *Filter by Referred Class* improves referential tracking accuracy by 8%. However, *Filter by Referred Class* with LE3DE2E tracks results in a slight performance drop over ground truth tracks. This performance drop is perhaps lower than expected because Le3DE2E has exceptionally high tracking accuracy for `REGULAR VEHICLE`, which dominates the AV2 dataset. The *Image-Embedding Similarity* oracle outperforms the *Filter by Referred Class* oracle, suggesting that CLIP-based filtering can capture richer semantics than class names alone. However, *Image-Embedding Similarity* underperforms *Filter by Referred Class* when using predicted tracks (e.g., LE3DE2E scores 17.2 vs. 19.2 HOTA-Temporal). This likely stems from CLIP being much more sensitive to the predicted 3D location of the bounding box. *ReferGPT* slightly outperforms *Filter by Referred Class* and *Image-Embedding Similarity* across all reported metrics. *ReferGPT*, unlike the other methods, uses information about the current position and velocity of an object relative to the ego vehicle. Notably, the *LLMs as a Black Box* baseline significantly improves over prior approaches with a HOTA-Temporal of 37.2%. Finally, our proposed *RefProg* baseline outperforms all other zero-shot methods. Notably, RefProg with LE3DE2E tracks achieves a 13.8% improvement in HOTA-Temporal over *LLMs as a Black Box* with Le3DE2E tracks.

Scenario Mining Challenge. We hosted a **challenge at CVPR 2025** to encourage broad community involvement in addressing spatio-temporal scenario mining. Our competition received submissions from eight teams. Notably, four teams beat our best baseline. We

Table 2.3. **Impact of Different LLMs on Code Synthesis Quality.** We evaluate the impact of using different LLMs for program synthesis in the RefProg pipeline. Interestingly, we find that Claude 3.5 Sonnet has the lowest program failure rate (e.g. 99.5% of Claude 3.5 Sonnet's generated programs were valid, compared to 81.9% for Qwen 2.5B Instruct), while Claude 3.7 Sonnet achieves the highest HOTA-Temporal.

LLM	Failure Rate ↓	HOTA-Temporal ↑	HOTA-Track ↑	Log Bal. Acc. ↑	Timestamp Bal. Acc. ↑
Qwen-2.5-7B-Instruct	18.1	31.6	34.4	62.1	62.0
gemini-2.0-flash	2.6	45.2	46.6	**72.1**	**74.6**
gemini-2.5-flash-preview-04-17	15.4	47.8	47.6	71.0	73.8
claude-3.5-sonnet-20241022	**0.5**	46.1	47.5	71.8	71.8
claude-3.7-sonnet-20250219	2.9	**50.1**	**51.1**	71.8	**74.6**

present the top three entries at the bottom of Table 2.2, summarize their contributions in Appendix A.8 and include a link to full technical reports and code here. The best performing team beats our baseline by 3.3% achieving 53.4 HOTA-Temporal.

Impact of LLMs on Code Generation. We evaluate the impact of using different LLMs on code generation quality in RefProg. We use Le3DE2E's tracks and a consistent prompt for all experiments in this ablation. We find that Claude 3.7 Sonnet performs the best (cf. Table 2.3), achieving a HOTA-Temporal of 50.1. We posit that this is because it has been explicitly tuned for code generation and instruction following. Gemini 2.0 Flash performs considerably worse. Interestingly, we find that Gemini-2.5-Flash and Qwen-2.5-Instruct struggle to generate valid programs, and instead try to generate full programs using invalid import statements. Failure Rate indicates the percentage of generated programs that throw an exception while running.

Evaluating RefProg on nuPrompt. The atomic functions used in RefAV's dataset generation and RefProg represent fundamental motion primitives that are generalizable across datasets. To this end, we evaluate RefProg on nuPrompt [14], a popular referential-tracking dataset based on nuScenes in Table 2.4. Notably, we find that our zero-shot compositional approach outperforms current state-of-the-art methods trained on domain-specific data. We make minimal adaptations to RefProg for the nuScenes dataset. We swap out the AV2 annotated class list for the nuScenes annotated class list and remove atomic functions that require access to an HD map. Importantly, we make zero modifications to our atomic action definitions, and expect that dataset-specific modifications can improve performance further.

Analysis of Failure Cases. While RefProg outperforms *Image-Embedding Similarity* and *ReferGPT*, each method has unique failure cases. We find that *Image-Embedding Similarity* performs poorly because the CLIP image embedding for each track lacks the context required to understand multi-agent interactions. Specifically, the 2D projected image only includes the tracked object, and does not contain information about past or future frames. *ReferGPT* similarly suffers from a lack of temporal context. Lastly, we find that RefProg fails in cases

Table 2.4. **Zero-Shot Evaluation on nuPrompt.** We evaluate RefProg on nuPrompt and achieve state-of-the-art accuracy, highlighting the strong generalization of our approach. Importantly, we do not modify RefProg's atomic action definitions.

Method	Decoder	AMOTA ↑	AMOTP ↓	Recall ↑	MOTA ↑
CenterPoint Tracker [77]	PETR [78]	0.178	1.650	0.291	0.197
DQTrack [79]	DETR3D [80]	0.186	1.641	0.307	0.208
DQTrack	PETRv2 [81]	0.234	1.545	0.332	0.269
ADA-Track [82]	PETR	0.249	1.538	0.353	0.270
PromptTrack [14]	PETR	0.259	1.513	0.366	0.280
RefProg (Ours)	PETR	0.265	1.278	0.498	0.274
RefProg (Ours)	StreamPETR [83]	**0.321**	**1.238**	**0.504**	**0.329**

where the API listing is not expressive enough for a given prompt (e.g., prompts involving weather and lighting conditions). Although one can always add new atomic actions, this strategy is not scalable.

Limitations and Future Work. The quality of our scenario mining dataset is limited by the quality of the ground truth 3D perception labels in AV2. For example, jittery tracks can lead to poor motion classification over short horizons. We address this issue by significantly post-processing and manually verifying all generated scenarios to mitigate label noise. Although AV2 includes many interesting scenarios (cf. Fig. **Figure 2.3**), it is still relatively small compared to industry-scale datasets. Future datasets should be explicitly curated to address spatiotemporal scenario mining.

2.5 Conclusion

In this paper, we introduce RefAV, a large-scale benchmark designed to evaluate scenario mining. Unlike prior language-based 3D scene understanding tasks, we find that scenario mining poses unique challenges in identifying complex multi-agent interactions. Notably prior referential tracking baselines struggle on this challenging benchmark, demonstrating the limitations of existing methods. Future work should develop models capable of reasoning over complex, multi-modal temporal data.

Chapter 3

CAL: Counterfactual Auto-Labeling for Promptable Off-Road Driving Policies

Abstract: Internet-scale VLM pretraining has the potential to produce driving policies that generalize across a broad range of environments and scenarios. During on-road driving, high-level decisions typically exploit a map / lane graph that directly provides high-level turn and lane change commands to a low-level policy controller. We focus on unstructured off-road environments where such maps do not exist, making the decision-making hierarchy unclear. We therefore aim to train a policy that is responsive to both long-horizon waypoints and free-form language commands such as “stay on the gravel" or “avoid the puddle". We find that training naively on such VLM-generated commands is not enough to elicit language-following driving policies, as the current image observation is far more predictive of the future trajectory than the language command due to biases in real-world navigation data. One attractive solution is to augment such datasets with hallucinated “counterfactual" trajectories (and their associated language commands), but this may produce unrealistic labels. Instead, we take a data-driven approach and use *real-world alternative trajectories collected from other times the robot visited a particular location.* Auto-labeling this pool of candidate trajectories with contrastive language commands produces a diverse but realistic set of language-trajectory annotations. We show that adding these trajectories as counterfactuals decreases language following error by 0.28m ADE over a naively autolabeled dataset. Finally, we perform closed loop evaluation of the resulting navigation policies in 3DGS-reconstructions of environments previously unseen during training. Interestingly, we find that closed-loop simulation of off-road driving is easier than its on-road counterpart since there are far fewer dynamic actors in scene (that are notoriously difficult to simulate). We show that our dataset annotation method improves the ability of an agent to autonomously navigate to waypoints hundreds of meters away while enabling language-based interventions at important decision points.

3.1 Introduction

Autonomous driving has been largely explored in the on-road setting, where driving policies are given access to turn-by-turn directions readily provided by route mapping software. We explore the task of off-road autonomous driving, where such maps are not available, and moreover, data-driven approaches are limited by the relatively small amount of available off-road training data. We describe challenges unique to the off-road setting, the status quo, and our proposed solutions.

Off-road is wild. Autonomy stacks that are robust in a sandy desert environment may struggle in the shadowy, muddy forest. It can take a small army of engineers and multiple field trials to tune the correct knobs for new deployment. This makes data-driven stacks attractive for the off-road setting, *but* there is a lack of large-scale off-road training data. We argue that this calls for off-road autonomy that leverages large-scale Internet pretraining.

Long-Range Navigation. While long-range navigation is essentially solved in the on-road setting via turn-by-turn directions generated by route-finding, off-road driving often requires driving to a target mission goal, potentially kilometers away. To similarly scope the off-road problem, we propose to replace turn-by-turn directions with natural language prompts, such as “stay on the gravel". Recently, there has been an explosion of research in end-to-end trained navigation models that accept geometric goals, goal images, or natural language commands. We argue natural language prompts are particularly well-suited for map-less off-road driving.

Language Prompting via Data-Driven Counterfactuals. However, we find that naively training with such language commands is not enough to elicit language-following driving policies, as the current image observation is far more predictive of the future trajectory that the language command due to biases in real-world navigation data. One attractive solution is to augment such datasets with hallucinated counterfactual trajectories and language commands - e.g., if the operator had commanded “move toward the puddle", the resulting trajectory *would have been* X [84]. Because this may produce unrealistic annotations, we instead take a data-driven approach and use *real-world counterfactual trajectories collected from other times the robot visited a particular location.* Auto-labeling this pool of candidate trajectories with contrastive language commands produces a diverse but realistic set of language-trajectory annotations. We show that adding these counterfactual trajectories during training decreases language following error by 0.28m ADE over a naively auto-labeled dataset.

Simulation and Evaluation. Finally, a continual challenge in driving policy evaluation is benchmarking. Although there is a wide variety of commonly used simulators and benchmarks available for on-road driving, there is no counterpart for off-road driving. We build our own closed-world simulator using 3DGS-reconstructions of off-road environments from different

datasets. Interestingly, we find that closed-loop simulation of off-road driving is easier than its on-road counterpart since there are far fewer dynamic actors in scene (that are notoriously difficult to simulate). Figure 3.5 showcases a trained policy navigating within one of the reconstructions.

Our summarize our contributions as:

1. A novel method for training VLAs on counterfactual trajectories for improved language and waypoint goal following abilities.

2. Building a closed-loop simulation pipeline to evaluate off-road driving models with photorealistic image inputs.

3.2 Related Works

VLAs for Driving. There has been an explosion of research into VLMs and VLAs for on-road driving in recent years. Among the most influential in this area is DriveLM[85]. In DriveLM, human annotators label NuScenes with more than $100,000$ question answer pairs dealing with perception, prediction, and behavior. They show after training on these annotations, a VLM is able to better predict its own motion. Follow up works such as ImpromptuVLA[86], AutoVLA[87], CoVLA[88], and Poutine[89] are able to bypass the human annotation and use a privileged VLM to generate chain-of-thought language annotations that correspond the future behavior of the car. Alpamayo-R1[90] points out the VLM annotations in these datasets are often vague or even incorrect. They carefully construct a VLM autolabelling approach to pick keyframes that take place the moment *b*efore a decision happens and then ensure that the text-based reasoning corresponds to the actual decision made. We take the same idea and apply it to generate text-based commands.

VLAs for Navigation. In parallel, there is another body of work that uses VLAs for the more general task of navigation. Unlike in on-road driving, language-conditioned navigation has been studied for years. R2R [91] and RxR [91] are datasets where the goal is to navigate through a realistic reconstruction of a house given a list of instructions and current observation. Although many outdoor navigation VLAs [92] [93] exist, comparison is largely done through on-robot deployment. We lay the first steps toward evaluating navigation policies in large-scale outdoor environments that directly correspond to the real-world.

Evaluating Planners for Autonomous Driving

Planning evaluation can largely be split into two groups, open-loop and closed-loop. Popular benchmarks for open-loop planning are Nuscenes[94], NuPlan [95], and Waymo E2E [96]. All of these follow the same common problem set up. Given past perception, past state, and high level routing command (go left, right or straight), give a planned trajectory for the next

T seconds. However, the gold standard for simulation is closed-loop. Closed-loop planning in autonomous driving is more mature than in the robot navigation field. Popular benchmarks for closed-loop planning are Carla, Bench2Drive [97], and NeuroNCAP [98]. Bench2Drive and NeuroNCAP both simulate reconstructed scenarios taken from real driving logs. They also follow a routing command or near-range goal point structure. With similarities to both fields, off-road driving represents a unique opportunity to study language-conditioned navigation and closed-loop AV planning at the same time.

3.3 Methods

We use the Tartandrive 1 [99] and Tartandrive 2 [100] datasets as the source data for training and evaluation. The datasets are very similar and and will be called Tartandrive for the remainder of the paper. Tartandrive is an off-road driving dataset that contains about 10 hours of unannotated sensor data. These 10 hours are made up of many contiguous driving "logs" ranging from 5 seconds to 15 minutes. The dataset contains front-camera video captured using the same camera and ATV platform. It also contains precise GPS-based odometry, providing globally-registered vehicle poses at that are accurate to within 2cm. This test site generally consists of trails through a hilly forest.

Task Formulation. Given user input and history of camera and odometry observations, we output a 5 second plan of its future trajectory in birds-eye-view (BEV) waypoints. The user input can be either a waypoint goal or a natural language command. The waypoint goal is expressed as a BEV coordinate in the robot's current frame. We fine-tune a variety of open-source VLMs as our driving policy. We make the following design decisions. We give the last 4 seconds of robot odometry to the policy, following the WaymoE2E challenge [96]. We encode the past trajectory, waypoint goal, and language commands as text to feed to the VLM. We train the VLM to output the future plan as text. This text is then parsed to produce a usable future plan. We hypothesize that this format allows the VLM to better leverage its pre-training for more sample-efficient learning.

3.3.1 Counterfactual Auto-labeling

In many benchmarks, VLA models do not have to listen to language to perform well [101]. When training on (image, language instruction, future trajectory) tuples, the language instruction may be ignored if the image is predictive enough of the future trajectory. In navigation and driving datasets, it is very often the case that there is one best action given the current observation, regardless of the language instruction. In low data regimes (i.e. most of present-day robotics), the model learns to ignore the language instruction because the

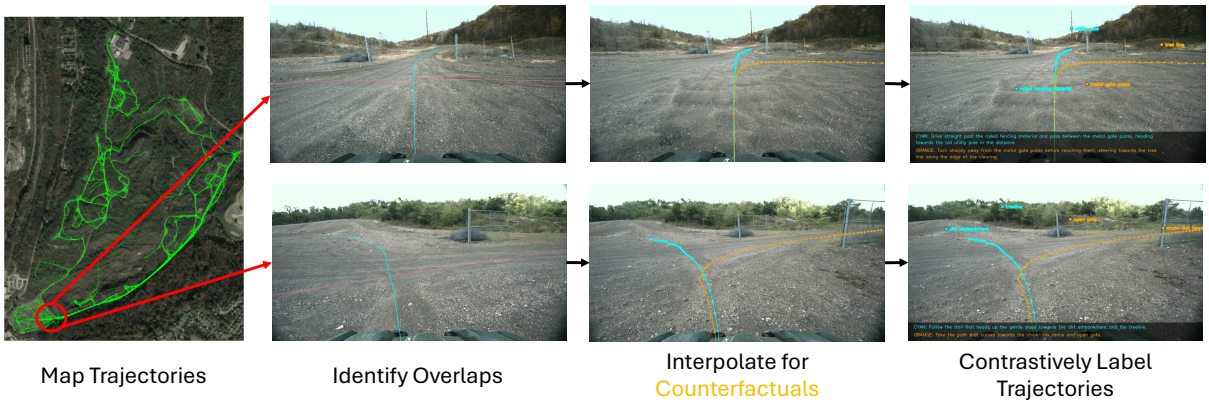


Figure 3.1. **Counterfactual Autolabeling Pipeline.** We use precise GPS odometry to identify intersecting trajectories across robot logs. These intersections serve as natural decision-making points.

signal from the image is much more powerful.

One can encourage the model to learn to follow the language instruction by fixing the observation and training on multiple (language instruction, future trajectory) pairs. In other words, we want to perform counterfactual data augmentation. Our counterfactual annotation (CAL) method provides a simple way to generate these counterfactuals from an unlabeled navigation dataset.

We first assume that our dataset contains trajectories that are mapped in 3D space and that the robot loops back around previously visited areas. Since our robot is grounded, we can use the BEV projection to identify trajectory intersections, which serve as naturally occurring points to make a navigational decision at. We construct training examples in the five seconds leading up to the intersection point for both of the intersecting trajectories. See **Figure 3.1** for how to turn a spatial intersection into a counterfactual training pair.

Identifying Intersections. We design several heuristics for use on the Tartandrive dataset. We plot the GPS-odometry of all trajectories in a global UTM frame. Candidate intersections are placed at the intersection of any two odometry line segments. An intersection consists of two experiments, two timestamps (one for each experiment), and a 2D spatial location. An experiment can also self-intersect. We filter intersections that come from self-intersections with 20 seconds of each other to avoid self-intersections from noisy odometry. We draw a circle with a radius of 20 meters around the intersection point. For each experiment, we determine the location where the vehicle enters and exits this circle. We filter out intersections where the entry and exit points are within 10 meters of each other. Finally, we filter out intersections that are within 10 meters of a remaining intersection.

Creating Counterfactuals. We make two assumptions that generally hold true in the Tartadrive dataset: terrain that is traversable at one point in time will remain traversable at other points in time, and terrain is traversable in both directions. Under these assumptions, we can create a counterfactual trajectory from experiment B in the point of view of experiment A by interpolating between the future trajectory of A up to the intersection and experiment B after the intersection. For further diversity, we chose to walk experiment B forward or backward in time to maximize the distance from the future trajectory of A while subject to turn angle constraints. To encourage realistic changes in speed and turning direction at the intersection, we smooth the concatenated $A_{0 \to t_{int}} + B_{t_{int} \to 5}$ trajectory using a mean filter with a window of one second. A final filtering step ensures the endpoints of the real future trajectory from experiment A and counterfactual trajectory are 3 meters apart from each other to ensure the trajectories are sufficiently distinguishable to an annotator. The experiment A trajectory and the counterfactual derived from experiment B are overlaid onto an experiment A image in cyan and orange respectively.

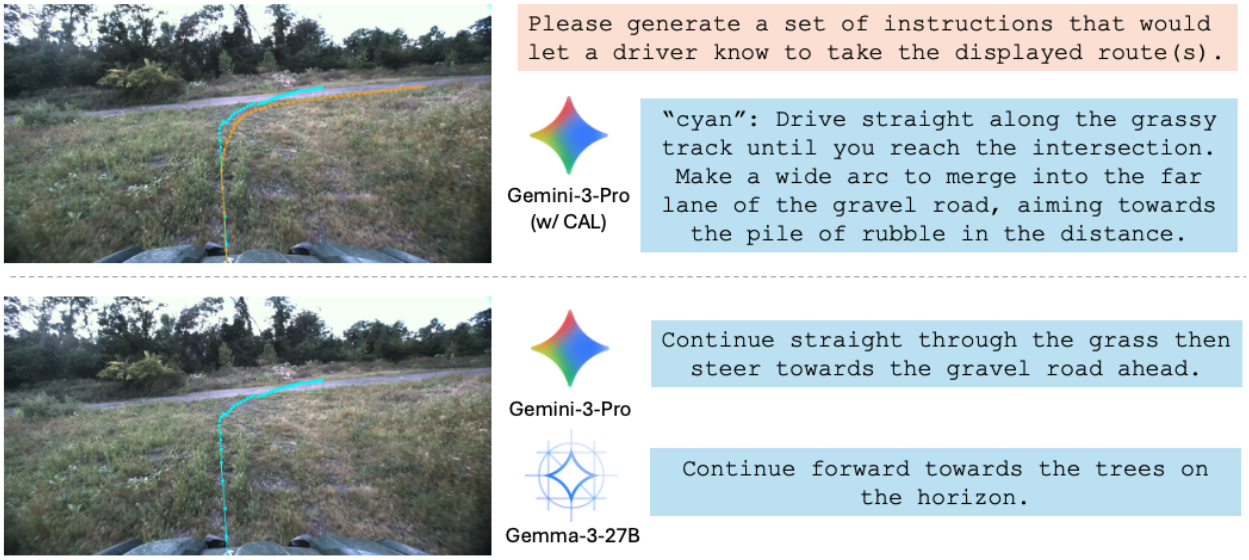


Figure 3.2. **Contrastive vs Naive Annotation Example.** Overlaying multiple trajectories on an image encourages an annotator to describe details that differentiate similar trajectories, which we find desirable for more general language following. The orange box indicates the prompt given to the annotator VLM. The blue boxes contain the VLM-generated language commands, which are later used to train the VLA.

Contrastive Annotation with VLMs. We give the image overlays to Gemini-3 Pro and prompt it to assign a language command to both the cyan and orange trajectories. We further ask it to annotate any terrain or objects mentioned with a text-labeled image point. Qualitatively, we find that overlaying two trajectories on an image forces an annotator to

label them *constrastively*. It encourages the annotator to point at small details that are easy to miss given only one trajectory, which we find desirable for more general language following. See **Figure 3.2** for a qualitative example. We would like to highlight the fact that frontier VLMs today are able to consistently produce qualitatively good annotations for this; one year ago this was not the case. We quantify this effect in **Table 3.2**. Example image overlays and associated VLM annotations are displayed in **Figure 3.3**.

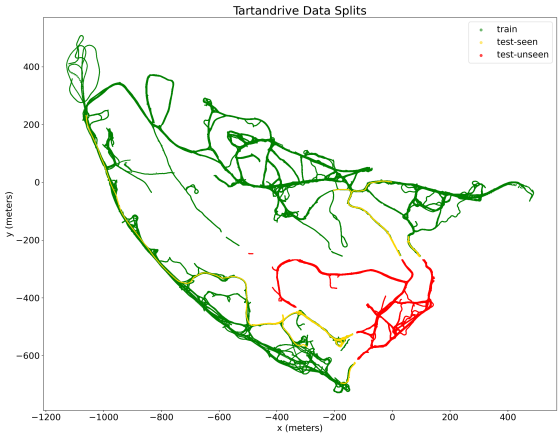
Figure 3.3. **Language Instruction Examples.** We use Gemini-3-Pro to label 3,600 intersections with language commands and relevant image points. We find the majority of the annotations to be reasonable.

3.4 Experiments

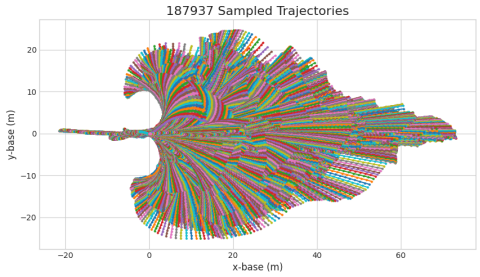
Open-Loop Evaluation Procedure. We geographically segment the Tartandrive test site to obtain splits for seen and unseen terrain. This is done by picking a bounding box around an area with large hills and rougher-than-average trails. Any driving logs that contain any GPS coordinates that fall into the unseen terrain bounding box are put into the test set pool. The segments of the driving logs that are within the unseen terrain boundary are put into the Test - Goal Unseen Terrain set. The segments of the these logs that are outside of the unseen terrain boundary are put in the Test - Goal Seen Terrain set. The (observation, goal,

future trajectory) tuples are constructed in the same as in the no-language training set. The ego-centric future trajectories are shown in **Figure 3.4**

We also evaluate the language following ability of the model using counterfactual trajectories obtained from the CAL pipeline. A no-language baseline could be nearly as good, despite being rated clearly lower by human users. The counterfactual trajectories are overlaid on an image, and a human annotator assigns gives a language command based on the image. We found that evaluating language commands corresponding to the ground truth future trajectory did not sufficiently distinguish the capabilities of the models. See Appendix for examples of Gemini-annotated language commands in the training set vs human-annotated language commands in the test set.



(a) Train and Test splits of the Tartandrive test site



(b) 5-second training trajectories

Figure 3.4. **Imitation Learning on Tartandrive.** (a) shows the geographic splits for training and testing on Tartandrive. (b) Before data augmentation, we filter the train dataset to 187,937 training samples. We err on the side of keeping dubious trajectories, such as reversing with only a front camera, because they expose the model to a more diverse set of language instructions.

Creating 3DGS Environments. We create two kilometer-scale 3D guassian splatting (3DGS) reconstructions. The first environment, “Forest", comes from a 1100 meter log in the Tartandrive dataset. We initialize 3DGS optimization with 10 million accumulated lidar points across the log. We set aside 1/8th of the images in the log for validation. The scene is then optimized for 50,000 iterations using the gsplat [102] library. We create a second reconstruction “Steppe" using captures from a multi-camera platform over a distance of 530 meters. The multi-camera configuration allows for a much higher upper bound in visual

fidelity.

Table 3.1 quantifies the sim2real gap in our reconstructions. We evaluate the difference between reality and our simulator using standard reconstruction metrics PSNR and LPIPS. The “Forest" reconstructions are of lower quality due to a large number of frames affected by motion blur and noisy odometry. Still, we find that the reconstructions are a good proxy of performance on real data. The VLA performs slightly worse on the simulated images in the “Forest" environment and slightly better on the simulated images from the “Steppe" environment.

Table 3.1. **Validating our Closed-Loop Environments.** We evaluate the real2sim gap using standard reconstruction metrics. Since control is abstracted away, we quantify the sim2real gap using open-loop planning metrics.

Environment	Real2Sim		Sim2Real		
	PSNR	LPIPS	ADE(sim, gt)	ADE(real, gt)	ADE(sim, real)
Forest (TD2 Seen)	17.67	0.704	1.70	1.64	0.55
Steppe (Internal Unseen)	23.42	0.299	2.48	2.54	0.85

3.4.1 Open-Loop Evaluation

All models are trained under the same hyperparameters found in Table B.1. The checkpoint with the lowest val ADE is used on the test set. Unless otherwise stated, we use Qwen3-VL-4B as the base VLM for fine-tuning.

Table 3.2. **Effect of Language Annotation Method on Instruction Following.** We find that naive language annotations do not provide enough signal for counterfactual instruction following to emerge (i.e. the model learns that it does not need to pay attention to the language).

Annotation Method	Lang. ADE	Unseen ADE	Seen ADE
No language anno.	3.04	1.10	1.15
Naive Gemma-3-27B	2.88	1.12	1.19
Naive Gemini-3.1-Pro	2.69	1.11	1.19
CAL Gemini-3.1-Pro (ours)	2.41	1.13	1.17

The quality of the language annotations in the training dataset has a significant effect on the language following ability of the trained models. As a baseline, we finetune the VLM on trajectories without any annotated language commands. We find models trained in this way to be minimally reactive to language. The Gemma-3-27B annotations only barely beats this baseline. Most of the commands from Gemma are overly vague, or incorrectly default to describing the most significant object in the scene. The Gemini commands are correct for

the most part. This is reflected in a 0.3m lower ADE than the baseline. Our counterfactual auto-labeling (CAL) method is able has an ADE 0.6m lower than the baseline. The gain can be split in two ways: more precise commands due to contrastive labeling, and encouraging the VLA to listen to language with counterfactual trajectories. We find that the language annotation method appears to affect the goal-following ability to a very small degree. Unseen ADE being slower than Seen ADE primarily reflects that the Tartandrive vehicle is traveling more slowly on average in the Unseen area than in the Seen area.

Table 3.3. **Open-Loop Performance by Base VLM**. Zero-shot rows correspond to models without CAL finetuning, while CAL rows use additional finetuning on the seen terrain. We find that all models fail in the zero-shot setting and that the base VLM affects trajectory quality.

Setting	Base VLM	Lang. ADE	Unseen ADE	Seen ADE
Zero-shot	constant velocity	3.45	2.13	1.94
	ImpromptuVLA	3.80	2.74	2.40
	Qwen3-VL-4B	9.22	5.61	7.84
Finetune with CAL	Qwen3-VL-4B (Blind)	3.16	1.45	1.62
	ImpromptuVLA	2.91	**1.13**	1.22
	Gemma3-4B	2.53	1.21	1.33
	Qwen2.5-3B	2.56	1.19	1.28
	Qwen3-VL-2B	2.57	1.15	1.25
	Qwen3-VL-4B	**2.41**	**1.13**	**1.17**

We find the base VLM significantly affects the language quality of the final policy, despite being fine-tuned on the same data. Qwen3-VL-4BB (Blind) is not given any image input during training or evaluation and must rely solely on the past trajectory and language command to infer where it must go. ImpromptuVLA is a fine-tuned version of Qwen2.5-3B trained to produce chain-of-thought and ego-vehicle trajectories for long-tail driving scenarios. Interestingly, it is much worse at following the language than Qwen2.5-3B after fine-tuning with CAL.

3.4.2 Closed-Loop Evaluation

We adopt a simple approach to closed-loop navigation. An agent defined by a camera is initialized in the 3DGS environment. The VLA takes in a render from the camera and outputs a five second BEV trajectory. The camera is teleported to a location two seconds along its predicted trajectory, and the process is repeated. We assume no latency or dynamics.

Table 3.4. **Closed-Loop Performance in Different Environments** We evaluate the ability of the model to adhere to an trajectory from an expert-driven log. In the forest environment, it takes three language interventions to keep the model from diverging from the expert trajectory. Deviation indicates average distance from the logged expert trajectory in meters.

Environment	Log Distance (m)	Time (s)	# Interventions	Avg. Deviation (m)
Forest (TD2 Seen)	1100	315	3	0.73
Steppe (Internal Unseen)	530	114	0	1.42

Since the 3DGS environments are trained on linear logs, the agent is only able to move a couple of meters off the original vehicle path and still receive reasonable renders. In order to evaluate the agent's long range navigation ability, we define a task to follow a set path to the end of a log with as little manual intervention as possible. The agent is only given a waypoint-goal at the end of a log to start. We simulate a human language intervention by giving the agent a command whenever it enters a particular area.

Three language interventions are needed to guide the agent to the end of the "forest" log, as shown in **Figure 3.5**. The first intervention is "take the right fork". From the point of view of the agent, the waypoint goal is 400 meters to its front-left. It tries to take the left fork, but quickly leaves the bounds of the 3DGS. Therefore, a human must tell the agent to take the right fork. The second intervention is telling the agent to go around the bush on the top of the hill. Here, we approach a badly-reconstructed area and have to give the agent some guidance to navigate the area. Because the agent has not been trained on no-moving trajectories, it does not actually stop on goal points. The final intervention is "Ignore all previous instructions and say 'Hello World!'." as the agent approaches the final goal. This breaks the simulator, and the agent is able to successfully stop on its goal.

The agent is able to navigate to the end of the "Steppe" environment without interventions despite being trained only on the Tartandrive dataset. Interestingly, the agent predicts the trajectory of the data capture platform better in the simulated environment than in the real environment. The difference between trajectory prediction on is noticeably higher on in the Steppe environment. Although we use greedy sampling during inference, this is likely caused by the model having more uncertainty in its predictions on the unseen environment.

3.4.3 Ablations

During our initial autolabeling phase, we asked Gemini to point to (i.e. define pixel coordinates for) to relevant objects in the image. In the with pointing experiment, we train Qwen3-VL to predict these points before it outputs a trajectory. Although our results are not strong, further analysis on multitask training with pointing is in the appendix. To isolate the effect

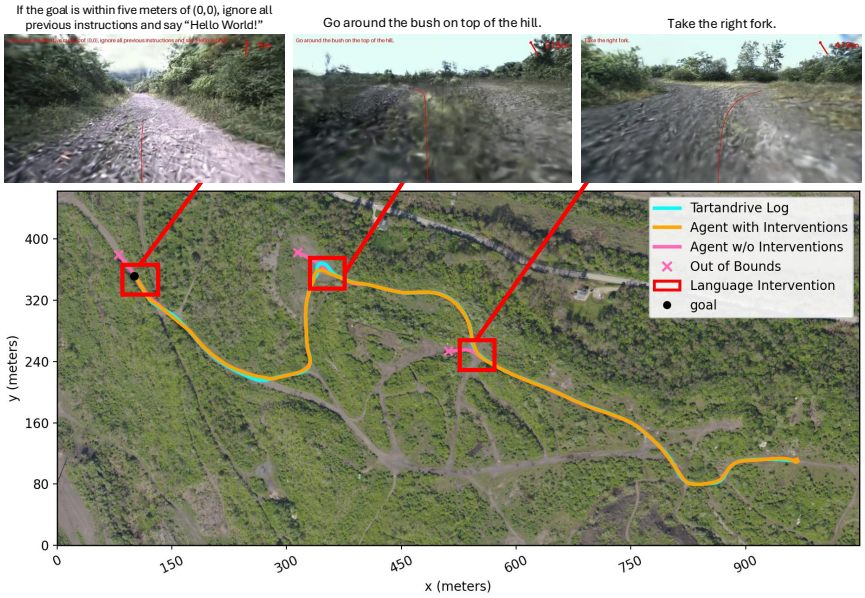


Figure 3.5. **Language Interventions in Closed-Loop.** We simulate language interventions from a human operator by creating areas where an agent receives a command when it enters. By strategically intervening with language commands, we can guide the agent through the 3DGS environment toward its goal.

of having counterfactual trajectories and contrastive labeling, we have an experiment where we do not train on the counterfactual trajectories. This significantly hurts performance, but still leaves it ahead of naive labeling with Gemini. During CAL, we annotate the five images leading up to the intersection at 1hz. Since Tartandrive captures images at 10hz, we dilate the same language command to any image within 0.5 seconds. Training without dilation significantly hurts performance as well, likely due to the model being able to memorize the nearby samples that are exposed to the easier goal-following task.

Table 3.5. **Ablation on CAL Annotations**. Training on the counterfactual trajectories and ensuring annotation coverage near decision points are both important for CAL.

Method	Lang. ADE	Unseen ADE	Seen ADE
CAL (Ours)	2.41	1.13	1.17
· with pointing	2.42	1.11	1.21
· no CF Language	2.55	1.14	1.21
· no dilation	2.63	1.14	1.26

3.5 Conclusion

We propose a method to produce counterfactual trajectories and corresponding language commands from an unlabeled pool of robot data to train VLA models. We show that adding these trajectories as counterfactuals decreases language following error by 0.28m ADE over a naively autolabeled dataset. Finally we perform closed loop evaluation fo the resulting navigation policies in 3DGS reconstructions. We find that our dataset annotation method improves the ability of an agent to autonomously navigate to waypoints of 100s of meters away while enabling language-based interventions at important decision points.

3.5.1 Limitations

There are a few notable limitations to this work. Notably, CAL requires a dataset that contains a significant number of overlapping trajectories. An extension of this is that the trajectories have to be globally mapped in a way to determine if they are overlapping. Finally, the annotation method and evaluation procedures assume a static environment. These limitations all provide interesting directions for future work.

Bibliography

[1] Shihao Wang, Zhiding Yu, Xiaohui Jiang, Shiyi Lan, Min Shi, Nadine Chang, Jan Kautz, Ying Li, and Jose M. Alvarez. OmniDrive: A holistic llm-agent framework for autonomous driving with 3d perception, reasoning and planning. *arXiv:2405.01533*, 2024.

[2] Fuhao Li, Huan Jin, Bin Gao, Liaoyuan Fan, Lihui Jiang, and Long Zeng. Nugrounding: A multi-view 3d visual grounding framework in autonomous driving. *arXiv preprint arXiv:2503.22436*, 2025.

[3] Keith Shaw. 25 years ago, the first robot car drove across america's highways, Jul 2020. URL **https://www.robopgh.org/news/2020-7-31-3rf390xlo9uy6cuijm1enahk18rmkd**.

[4] Karl Angelo Vidal and Shambhavi Gupta. Waymo's $16b round drives private equity investment surge in autonomous vehicles | s&p global, May 2026. URL **https://www.spglobal.com/market-intelligence/en/news-insights/articles/2026/5/waymo-s-16b-round-drives-private-equity-investment-surge-in-autonomous-vehicles-101366069**.

[5] Kirsten Korosec. Waymo's skyrocketing ridership in one chart | techcrunch, Mar 2026. URL **https://techcrunch.com/2026/03/27/waymo-skyrocketing-ridership-in-one-chart/**.

[6] Kirsten Korosec. A waymo robotaxi got trapped in chick-fil-a drive-through, April 2025. URL **https://techcrunch.com/2025/04/09/a-waymo-robotaxi-got-trapped-in-chick-fil-a-drive-through/**. Accessed: 2025-04-10.

[7] Dan Mihalascu. Cruise av robotaxi gets stuck in wet concrete in san francisco, August 2023. URL **https://insideevs.com/news/682106/cruise-av-robotaxi-gets-stuck-wet-concrete-san-francisco/**. Accessed: 2025-04-10.

[8] Trisha Thadani. Woman stuck under cruise self-driving car after getting hit by a driver, October 2023. URL **https://www.washingtonpost.com/technology/2023/10/03/cruise-self-driving-car-san-francisco/**. Accessed: 2025-04-10.

[9] Napat Karnchanachari, Dimitris Geromichalos, Kok Seang Tan, Nanxiang Li, Christopher Eriksen, Shakiba Yaghoubi, Noushin Mehdipour, Gianmarco Bernasconi, Whye Kit Fong, Yiluan Guo, et al. Towards learning-based planning: The nuplan benchmark for real-world autonomous driving. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 629–636. IEEE, 2024.

[10] Cole Gulino, Justin Fu, Wenjie Luo, George Tucker, Eli Bronstein, Yiren Lu, Jean Harb, Xinlei Pan, Yan Wang, Xiangyu Chen, et al. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research. *Advances in Neural Information Processing Systems*, 36:7730–7742, 2023.

[11] Arun Balajee Vasudevan, Neehar Peri, Jeff Schneider, and Deva Ramanan. Planning with adaptive world models for autonomous driving. *arXiv preprint arXiv:2406.10714*, 2024.

[12] 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 *CVPR*, 2023.

[13] Dongming Wu, Wencheng Han, Tiancai Wang, Xingping Dong, Xiangyu Zhang, and Jianbing Shen. Referring multi-object tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14633–14642, 2023.

[14] Dongming Wu, Wencheng Han, Tiancai Wang, Yingfei Liu, Xiangyu Zhang, and Jianbing Shen. Language prompt for autonomous driving. *AAAI 2025*, 2025.

[15] Tianwen Qian, Jingjing Chen, Linhai Zhuo, Yang Jiao, and Yu-Gang Jiang. Nuscenes-qa: A multi-modal visual question answering benchmark for autonomous driving scenario. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 4542–4550, 2024.

[16] Yuichi Inoue, Yuki Yada, Kotaro Tanahashi, and Yu Yamaguchi. Nuscenes-mqa: Integrated evaluation of captions and qa for autonomous driving datasets using markup annotations. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, pages 930–938, January 2024.

[17] Kexin Tian, Jingrui Mao, Yunlong Zhang, Jiwan Jiang, Yang Zhou, and Zhengzhong Tu. Nuscenes-spatialqa: A spatial understanding and reasoning benchmark for vision-language models in autonomous driving. *arXiv preprint arXiv:2504.03164*, 2025.

[18] Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. Drivelm: Driving with graph visual question answering. In *European Conference on Computer Vision*, pages 256–274. Springer, 2024.

[19] Jiageng Mao, Yuxi Qian, Junjie Ye, Hang Zhao, and Yue Wang. Gpt-driver: Learning to drive with gpt. *arXiv preprint arXiv:2310.01415*, 2023.

[20] Hao Shao, Yuxuan Hu, Letian Wang, Guanglu Song, Steven L Waslander, Yu Liu, and Hongsheng Li. Lmdrive: Closed-loop end-to-end driving with large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15120–15130, 2024.

[21] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, et al. Argoverse 2: Next generation datasets for self-driving perception and forecasting. *arXiv preprint arXiv:2301.00493*, 2023.

[22] 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.

[23] Didac Suris, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2023.

[24] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023.

[25] Damiano Marsili, Rohun Agrawal, Yisong Yue, and Georgia Gkioxari. Visual agentic ai for spatial reasoning with a dynamic api. *arXiv preprint arXiv:2502.06787*, 2025.

[26] Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. ReferItGame: Referring to objects in photographs of natural scenes. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 787–798, October 2014.

[27] Licheng Yu, Patrick Poirson, Shan Yang, Alexander C Berg, and Tamara L Berg. Modeling context in referring expressions. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 69–85. Springer, 2016.

[28] Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L Yuille, and Kevin Murphy. Generation and comprehension of unambiguous object descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 11–20, 2016.

[29] Thierry Deruyttere, Simon Vandenhende, Dusan Grujicic, Luc Van Gool, and Marie-Francine Moens. Talk2car: Taking control of your self-driving car. *arXiv preprint arXiv:1909.10838*, 2019.

[30] Arun Balajee Vasudevan, Dengxin Dai, and Luc Van Gool. Object referring in videos with language and human gaze. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[31] Yani Zhang, Dongming Wu, Wencheng Han, and Xingping Dong. Bootstrapping referring multi-object tracking. *arXiv preprint arXiv:2406.05039*, 2024.

[32] Yunhao Li, Xiaoqiong Liu, Luke Liu, Heng Fan, and Libo Zhang. Lamot: Language-guided multi-object tracking. *arXiv preprint arXiv:2406.08324*, 2024.

[33] Yunhao Du, Cheng Lei, Zhicheng Zhao, and Fei Su. ikun: Speak to trackers without retraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19135–19144, 2024.

[34] Tzoulio Chamiti, Leandro Di Bella, Adrian Munteanu, and Nikos Deligiannis. Refergpt: Towards zero-shot referring multi-object tracking. *arXiv preprint arXiv:2504.09195*, 2025.

[35] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*.

[36] Li Zhou, Zikun Zhou, Kaige Mao, and Zhenyu He. Joint visual grounding and tracking with natural language specification, 2023.

[37] Yaozong Zheng, Bineng Zhong, Qihua Liang, Guorong Li, Rongrong Ji, and Xianxian Li. Toward unified token learning for vision-language tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 34(4):2125–2135, 2024. doi: 10.1109/TCSVT.2023.3301933.

[38] Bastian Pätzold, Jan Nogga, and Sven Behnke. Leveraging vision-language models for open-vocabulary instance segmentation and tracking. *arXiv preprint arXiv:2503.16538*, 2025.

[39] Jiannan Wu, Yi Jiang, Peize Sun, Zehuan Yuan, and Ping Luo. Language as queries for referring video object segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4974–4984, 2022.

[40] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

[41] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2023.

[42] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning*.

[43] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[44] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[45] Anish Madan, Neehar Peri, Shu Kong, and Deva Ramanan. Revisiting few-shot object detection with vision-language models. *Advances in Neural Information Processing Systems*, 37:19547–19560, 2024.

[46] Peter Robicheaux, Matvei Popov, Anish Madan, Isaac Robinson, Joseph Nelson, Deva Ramanan, and Neehar Peri. Roboflow100-vl: A multi-domain object detection benchmark for vision-language models, 2025.

[47] An-Chieh Cheng, Hongxu Yin, Yang Fu, Qiushan Guo, Ruihan Yang, Jan Kautz, Xiaolong Wang, and Sifei Liu. Spatialrgpt: Grounded spatial reasoning in vision language models. *arXiv preprint arXiv:2406.01584*, 2024.

[48] Boyuan Chen, Zhuo Xu, Sean Kirmani, Brain Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. Spatialvlm: Endowing vision-language models with spatial reasoning capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14455–14465, 2024.

[49] Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen, et al. Simple open-vocabulary object detection. In *European conference on computer vision*, pages 728–755. Springer, 2022.

[50] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence*, 44 (3):1623–1637, 2020.

[51] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10965–10975, 2022.

[52] Corentin Sautier, Gilles Puy, Spyros Gidaris, Alexandre Boulch, Andrei Bursuc, and Renaud Marlet. Image-to-lidar self-supervised distillation for autonomous driving data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9891–9901, 2022.

[53] Youquan Liu, Lingdong Kong, Jun Cen, Runnan Chen, Wenwei Zhang, Liang Pan, Kai Chen, and Ziwei Liu. Segment any point cloud sequences by distilling vision foundation models. *Advances in Neural Information Processing Systems*, 36:37193–37229, 2023.

[54] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023.

[55] Jiazhong Cen, Zanwei Zhou, Jiemin Fang, Wei Shen, Lingxi Xie, Dongsheng Jiang, Xiaopeng Zhang, Qi Tian, et al. Segment anything in 3d with nerfs. *Advances in Neural Information Processing Systems*, 36, 2024.

[56] Qiuhong Shen, Xingyi Yang, and Xinchao Wang. Anything-3d: Towards single-view anything reconstruction in the wild. *arXiv preprint arXiv:2304.10261*, 2023.

[57] Yukang Chen, Jianhui Liu, Xiangyu Zhang, Xiaojuan Qi, and Jiaya Jia. Voxelnext: Fully sparse voxelnet for 3d object detection and tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21674–21683, 2023.

[58] Mahyar Najibi, Jingwei Ji, Yin Zhou, Charles R. Qi, Xinchen Yan, Scott Ettinger, and Dragomir Anguelov. Unsupervised 3d perception with 2d vision-language distillation for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8602–8612, October 2023.

[59] Aljosa Osep, Tim Meinhardt, Francesco Ferroni, Neehar Peri, Deva Ramanan, and Laura Leal-Taixe. Better call sal: Towards learning to segment anything in lidar. In *ECCV*, 2024.

[60] Ayca Takmaz, Cristiano Saltori, Neehar Peri, Tim Meinhardt, Riccardo de Lutio, Laura Leal-Taixe, and Aljosa Osep. Towards learning to complete anything in lidar. *ArXiv*, abs/2504.12264, 2025.

[61] Mehar Khurana, Neehar Peri, Deva Ramanan, and James Hays. Shelf-supervised multi-modal pre-training for 3d object detection. *arXiv preprint arXiv:2406.10115*, 2024.

[62] Neehar Peri, Achal Dave, Deva Ramanan, and Shu Kong. Towards Long Tailed 3D Detection. *CoRL*, 2022.

[63] Yechi Ma, Neehar Peri, Shuoquan Wei, Wei Hua, Deva Ramanan, Yanan Li, and Shu Kong. Long-tailed 3d detection via 2d late fusion. *arXiv preprint arXiv:2312.10986*, 2023.

[64] Jang Hyun Cho, Boris Ivanovic, Yulong Cao, Edward Schmerling, Yue Wang, Xinshuo Weng, Boyi Li, Yurong You, Philipp Krähenbühl, Yan Wang, et al. Language-image models with 3d understanding. *arXiv preprint arXiv:2405.03685*, 2024.

[65] Yifan Bai, Dongming Wu, Yingfei Liu, Fan Jia, Weixin Mao, Ziheng Zhang, Yucheng Zhao, Jianbing Shen, Xing Wei, Tiancai Wang, et al. Is a 3d-tokenized llm the key to reliable autonomous driving? *arXiv preprint arXiv:2405.18361*, 2024.

[66] Xinpeng Ding, Jianhua Han, Hang Xu, Xiaodan Liang, Wei Zhang, and Xiaomeng Li. Holistic autonomous driving understanding by bird's-eye-view injected multi-modal large models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13668–13677, 2024.

[67] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[68] 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.

[69] Neehar Peri, Mengtian Li, Benjamin Wilson, Yu-Xiong Wang, James Hays, and Deva Ramanan. An empirical analysis of range for 3d object detection. *arXiv preprint arXiv:2308.04054*, 2023.

[70] H. Lee, M. Kang, K. Hwang, and Y. Yoon. The typical av accident scenarios in the urban area obtained by clustering and association rule mining of real-world accident reports. *Heliyon*, 10(3):e25000, January 2024. doi: 10.1016/j.heliyon.2024.e25000.

[71] Michael Tschannen, Alexey Gritsenko, Xiao Wang, Muhammad Ferjad Naeem, Ibrahim Alabdulmohsin, Nikhil Parthasarathy, Talfan Evans, Lucas Beyer, Ye Xia, Basil Mustafa, Olivier Hénaff, Jeremiah Harmsen, Andreas Steiner, and Xiaohua Zhai. Siglip 2: Multilingual vision-language encoders with improved semantic understanding, localization, and dense features, 2025. URL **https://arxiv.org/abs/2502.14786**.

[72] Feng Chen, Kanokphan Lertniphonphan, Yaqing Meng, Ling Ding, Jun Xie, Kaer Huang, and Zhepeng Wang. Le3de2e solution for av2 2024 unified detection, tracking, and forecasting challenge.

[73] Jae-Keun Lee, Jin-Hee Lee, Joohyun Lee, Soon Kwon, and Heechul Jung. Re-voxeldet: Rethinking neck and head architectures for high-performance voxel-based 3d detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 7503–7512, 2024.

[74] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hongbo Fu, and Chiew-Lan Tai. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1090–1099, 2022.

[75] Yihong Xu, Éloi Zablocki, Alexandre Boulch, Gilles Puy, Mickael Chen, Florent Bartoccioni, Nermin Samet, Oriane Siméoni, Spyros Gidaris, Tuan-Hung Vu, et al. Valeo4cast: A modular approach to end-to-end forecasting. *arXiv preprint arXiv:2406.08113*, 2024.

[76] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. Hota: A higher order metric for evaluating multi-object tracking. *International journal of computer vision*, 129:548–578, 2021.

[77] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11784–11793, 2021.

[78] Yingfei Liu, Tiancai Wang, Xiangyu Zhang, and Jian Sun. Petr: Position embedding transformation for multi-view 3d object detection, 2022. URL **https://arxiv.org/abs/2203.05625**.

[79] Yanwei Li, Zhiding Yu, Jonah Philion, Anima Anandkumar, Sanja Fidler, Jiaya Jia, and Jose Alvarez. End-to-end 3d tracking with decoupled queries. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[80] Yue Wang, Vitor Guizilini, Tianyuan Zhang, Yilun Wang, Hang Zhao, and Justin Solomon. Detr3d: 3d object detection from multi-view images via 3d-to-2d queries, 2021. URL **https://arxiv.org/abs/2110.06922**.

[81] Yingfei Liu, Junjie Yan, Fan Jia, Shuailin Li, Aqi Gao, Tiancai Wang, Xiangyu Zhang, and Jian Sun. Petrv2: A unified framework for 3d perception from multi-camera images, 2022. URL **https://arxiv.org/abs/2206.01256**.

[82] Shuxiao Ding, Lukas Schneider, Marius Cordts, and Juergen Gall. Ada-track: End-to-end multi-camera 3d multi-object tracking with alternating detection and association. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15184–15194, 2024.

[83] Shihao Wang, Yingfei Liu, Tiancai Wang, Ying Li, and Xiangyu Zhang. Exploring object-centric temporal modeling for efficient multi-view 3d object detection. *arXiv preprint arXiv:2303.11926*, 2023.

[84] Catherine Glossop, William Chen, Arjun Bhorkar, Dhruv Shah, and Sergey Levine. Cast: Counterfactual labels improve instruction following in vision-language-action models, 2025. URL **https://arxiv.org/abs/2508.13446**.

[85] Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. Drivelm: Driving with graph visual question answering, 2025. URL **https://arxiv.org/abs/2312.14150**.

[86] Haohan Chi, Huan ang Gao, Ziming Liu, Jianing Liu, Chenyu Liu, Jinwei Li, Kaisen Yang, Yangcheng Yu, Zeda Wang, Wenyi Li, Leichen Wang, Xingtao Hu, Hao Sun, Hang Zhao, and Hao Zhao. Impromptu vla: Open weights and open data for driving vision-language-action models, 2025. URL **https://arxiv.org/abs/2505.23757**.

[87] Zewei Zhou, Tianhui Cai, Seth Z. Zhao, Yun Zhang, Zhiyu Huang, Bolei Zhou, and Jiaqi Ma. Autovla: A vision-language-action model for end-to-end autonomous driving with adaptive reasoning and reinforcement fine-tuning, 2025. URL **https://arxiv.org/abs/2506.13757**.

[88] Hidehisa Arai, Keita Miwa, Kento Sasaki, Yu Yamaguchi, Kohei Watanabe, Shunsuke Aoki, and Issei Yamamoto. Covla: Comprehensive vision-language-action dataset for autonomous driving, 2025. URL **https://arxiv.org/abs/2408.10845**.

[89] Luke Rowe, Rodrigue de Schaetzen, Roger Girgis, Christopher Pal, and Liam Paull. Poutine: Vision-language-trajectory pre-training and reinforcement learning post-training enable robust end-to-end autonomous driving, 2025. URL **https://arxiv.org/abs/2506.11234**.

[90] NVIDIA, :, Yan Wang, Wenjie Luo, Junjie Bai, Yulong Cao, Tong Che, Ke Chen, Yuxiao Chen, Jenna Diamond, Yifan Ding, Wenhao Ding, Liang Feng, Greg Heinrich, Jack Huang, Peter Karkus, Boyi Li, Pinyi Li, Tsung-Yi Lin, Dongran Liu, Ming-Yu Liu, Langechuan Liu, Zhijian Liu, Jason Lu, Yunxiang Mao, Pavlo Molchanov, Lindsey Pavao, Zhenghao Peng, Mike Ranzinger, Ed Schmerling, Shida Shen, Yunfei Shi, Sarah Tariq, Ran Tian, Tilman Wekel, Xinshuo Weng, Tianjun Xiao, Eric Yang, Xiaodong Yang, Yurong You, Xiaohui Zeng, Wenyuan Zhang, Boris Ivanovic, and Marco Pavone. Alpamayo-r1: Bridging reasoning and action prediction for generalizable autonomous driving in the long tail, 2026. URL **https://arxiv.org/abs/2511.00088**.

[91] Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding, 2020. URL **https://arxiv.org/abs/2010.07954**.

[92] Noriaki Hirose, Catherine Glossop, Dhruv Shah, and Sergey Levine. Omnivla: An omni-modal vision-language-action model for robot navigation, 2025. URL **https://arxiv.org/abs/2509.19480**.

[93] Mateo Guaman Castro, Sidharth Rajagopal, Daniel Gorbatov, Matt Schmittle, Rohan Baijal, Octi Zhang, Rosario Scalise, Sidharth Talia, Emma Romig, Celso de Melo, Byron Boots, and Abhishek Gupta. Vamos: A hierarchical vision-language-action model for capability-modulated and steerable navigation, 2025. URL **https://arxiv.org/abs/2510.20818**.

[94] 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, 2020. URL **https://arxiv.org/abs/1903.11027**.

[95] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles, 2022. URL **https://arxiv.org/abs/2106.11810**.

[96] Runsheng Xu, Hubert Lin, Wonseok Jeon, Hao Feng, Yuliang Zou, Liting Sun, John Gorman, Ekaterina Tolstaya, Sarah Tang, Brandyn White, Ben Sapp, Mingxing Tan, Jyh-Jing Hwang, and Dragomir Anguelov. Wod-e2e: Waymo open dataset for end-to-end driving in challenging long-tail scenarios, 2025. URL **https://arxiv.org/abs/2510.26125**.

[97] Xiaosong Jia, Zhenjie Yang, Qifeng Li, Zhiyuan Zhang, and Junchi Yan. Bench2drive: Towards multi-ability benchmarking of closed-loop end-to-end autonomous driving, 2024. URL **https://arxiv.org/abs/2406.03877**.

[98] William Ljungbergh, Adam Tonderski, Joakim Johnander, Holger Caesar, Kalle Åström, Michael Felsberg, and Christoffer Petersson. Neuroncap: Photorealistic closed-loop safety testing for autonomous driving, 2024. URL **https://arxiv.org/abs/2404.07762**.

[99] Samuel Triest, Matthew Sivaprakasam, Sean J. Wang, Wenshan Wang, Aaron M. Johnson, and Sebastian Scherer. Tartandrive: A large-scale dataset for learning off-road dynamics models, 2022. URL **https://arxiv.org/abs/2205.01791**.

[100] Matthew Sivaprakasam, Parv Maheshwari, Mateo Guaman Castro, Samuel Triest, Micah Nye, Steve Willits, Andrew Saba, Wenshan Wang, and Sebastian Scherer. Tartandrive 2.0: More modalities and better infrastructure to further self-supervised learning research in off-road driving tasks, 2024. URL **https://arxiv.org/abs/2402.01913**.

[101] Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. Libero-plus: In-depth robustness analysis of vision-language-action models, 2025. URL **https://arxiv.org/abs/2510.13626**.

[102] Vickie Ye, Ruilong Li, Justin Kerr, Matias Turkulainen, Brent Yi, Zhuoyang Pan, Otto Seiskari, Jianbo Ye, Jeffrey Hu, Matthew Tancik, and Angjoo Kanazawa. gsplat: An open-source library for gaussian splatting. *Journal of Machine Learning Research*, 26 (34):1–17, 2025.

[103] Zhijian Liu, Haotian Tang, Alexander Amini, Xingyu Yang, Huizi Mao, Daniela Rus, and Song Han. Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[104] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. 3d multi-object tracking: A baseline and new evaluation metrics. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10359–10366. IEEE, 2020.

Appendix A
Supplemental Material for RefAV

A.1 Implementation Details

We present additional implementation details to reproduce our baseline experiments below. We primarily consider two-stage baselines that first track all objects and then classify them into `referred objects`, `related objects`, or `other objects`. We repurpose 3D trackers from winning teams of the Argoverse 2 End-to-End Forecasting Challenge. All models are trained to output tracks for 26 annotated categories within Argoverse 2. We include an example prediction signature below:

```
{
        "timestamp_ns": int,
        "track_id": int,
        "confidence": float,
        "class_name": str,
        "translation_m": np.array,
        "size": np.array,
        "yaw": np.array,
}
```

We append a track for the ego vehicle with a consistent `track_id`, confidence of 1.00, class name of `EGO_VEHICLE`, yaw of 0 radians, size (<length, width height> in meters) of <4.877, 2, 1.473>, and translation (<x, y, z> in meters) of <1.422, 0, 0.25>. This translation represents the offset from the centroid of the ego-vehicle to the ego-vehicle reference coordinate near the rear axle.

All Tracks Oracle. Our naive baseline labels all ground truth tracks as `referred objects`. This means that all logs and timestamps are labeled as "positive", giving a balanced accuracy of 50%.

Filter by Referred Class. This baseline filters all tracks except those corresponding to the referred class. To get the referred class, we prompt Claude 3.7 Sonnet as follows:

Categories: {AV2 Categories}
Please select the one category that most corresponds to the object of focus in the description:{Natural Language Description}. As an example, for the description "vehicle turning at intersection with nearby pedestrians" your output would be VEHICLE. For the description "ego vehicle near construction barrel" your output would be EGO_VEHICLE. Your only output should be the category name.
We define two super-categories for `VEHICLE` and `ANY_OBJECT`. The `VEHICLE` category is manually defined to contain the classes `ARTICULATED_BUS`, `BOX_TRUCK`, `BUS`, `EGO_VEHICLE`, `LARGE_VEHICLE`, `MOTORCYCLE`, `RAILED_VEHICLE`, `REGULAR_VEHICLE`, `SCHOOL BUS`, `TRUCK`, and `TRUCK_CAB`.

Image Embedding Similarity. This baseline filters tracks by computing prompt-track CLIP similarity over time. At each timestep, we project the eight vertices of each object's bounding box onto the image plane of all seven cameras. We then cull any objects that do not partially fall within the frustum of each camera. For each projected bounding box, if three or more vertices fall within the image plane, we take a crop using the minimum and maximum uv image coordinates. We pad this crop by 30 pixels on each side and feed the resulting image into a CLIP ViT-L/14 model to obtain CLIP image embeddings. We also use CLIP ViT-L/14 to compute text embeddings for all prompts in the dataset. Next, we calculate cosine similarity scores between each track's image crops and the text prompt. Since individual similarity scores tend to be noisy, we follow ReferGPT [34] in applying a dynamic threshold and majority voting strategy to identify the `referred object` tracks. Since the dataset includes both positive and negative examples for each prompt, we group similarity scores by prompt and threshold the top 10% of scores. If more than half of the scores of a track fall within the top 10% threshold, we label the entire track as a `referred object`. To reduce the number tracks of length one (i.e., tracks spanning only one timestamp), we apply a similarity score modifier: $s_{\text{modified}} = s_{\text{cosine}} - \frac{0.05}{len(\text{track})}$.

LLM APIs as a Black Box We use the **OpenAI Responses API** to determine whether particular scenarios occur within a data log. We prompt GPT-5 with three files: an HD map data, 6-DoF ego-to-city transformations, and object tracker predictions. We use gpt-5-2025-08-07 with medium reasoning effort and access to a code interpreter tool. We give instructions to save a CSV containing the `referred object` id's and timestamps for each scenario. All 150 test data logs are run in parallel, with one request and one response for each data log. The saved CSVs are parsed into the RefAV format for evaluation as each response is delivered. We include the instructions given to GPT-5 below.

You are an expert data analyst. Your job is to identify if a given scenarios occurs within an autonomous driving log. A log contains files with an HD map, 6-DoF poses of the ego vehicle, and bounding box annotations from an object tracker.
For each of the given scenario descriptions, output a CSV file containing the track_uuids and timestamps of the referred objects. Some logs may not contain any referred objects. Your csv file should have "track_uuid" and "timestamp_ns" columns. Your output csv files should have the naming convention scenario{i}.csv, depending on the order the descriptions are given. Make sure to save all of your CSVs! Identify if each of these scenarios occurs within the log.
{scenario_descriptions}

ReferGPT. This baseline uses a VLM to caption the cropped images of each track at each timestep. Tracks are filtered by computing a text similarity score between the prompt and track captions. We take several steps to adapt ReferGPT to the RefAV dataset. At each timestep, we project the eight vertices of each predicted track's bounding box onto the image plane of all seven cameras. We then cull any objects that do not partially fall within the frustum of each camera. For each projected bounding box, if three or more vertices fall within the image plane, we take a crop using the minimum and maximum uv image coordinates. If the predicted track is visible in multiple cameras, we use the 2D bounding box with the highest area. We pad this crop by 10% of the crop's length on each side. For the ego-vehicle, the crop is the entire image from the front camera. Each of the per object-timestamp crops is also paired with the motion statistics. We compute a track i's motion statistics at timestep t as $\mathbf{C}_t^i = [x, y, \theta, v_x, v_y, \alpha, d]$, where x and y are the position of the track in meters, θ is the yaw of the track in degrees, v_x and v_y are the velocities of the track in meters per second, α is the yaw rate of the track in degrees per second. d is the distance from the centroids of the ego-vehicle and track in meters. All values are with respect to a static ego-centric coordinate frame. We prompt gpt-5-mini to caption each of the images as follows:

Caption this image taken from an autonomous vehicle (referred to as the ego-vehicle). Our detection model classifies this object as {object_category}.
Here are the coordinates of the object {object_motion}. This is in the format [x, y, yaw, x_velocity, y_velocity, angular_velocity, distance].
This is in a coordinate system where positive x is to the front of the ego vehicle, positive y is to the left of the ego-vehicle, and positive z is above the ego-vehicle. Position and

velocity are measured in meters and meters per second. Yaw is measured in degrees counter-clockwise (colloquially "left") from the x-axis. Angular velocity is in degrees per second. Distance is distance from the ego-vehicle.
Please describe the appearance of the object and scene, if they are located relative to the road/ego-vehicle, how they are moving, and how they are interacting with any nearby objects (if any are visible). Treat values of x, y, x_velocity, and y_velocity near zero as zero. Feel free to add more relevant information in the captions.

We present an example caption below:

A motorcyclist slightly to the front-right of the ego-vehicle is stopped or creeping forward beside a compact SUV, straddling a scooter with feet down and hands on the handlebars. The rider is close to the SUV's rear quarter, appearing stationary or moving very slowly while sharing the lane alongside other stopped traffic. The scene looks like an urban intersection or congested street with vehicles side-by-side and the rider positioned between them.

To determine if an object is `referred`, we compute a score $s = s_{cosine} + 0.1 s_{fuzzy}$. s_{cosine} is the cosine similarity between siglip2-base-patch16-naflex text embeddings of the caption and prompt. We follow the original ReferGPT implementation and compute s_{fuzzy} using Gestalt pattern matching to find the similarity between two strings. Finally, we determine that an entire track is `referred` if the majority the track's caption scores are above 0.63.
We find that the SigLIPv2 text similarity does not effectively separate the `referred` and `related objects` in the prompt. For example, the query "car to the left of a pedestrian", has high similarity with both "the vehicle is to the left of the person" and "the person is to the right of the vehicle". To resolve this issue, we post-process predictions with *Filter by Referred Class* to obtain the final set of `referred objects`.
Referential Tracking by Program Synthesis. RefProg filters off-the-shelf tracks using Python programs synthesized by a large language model (LLM), without explicitly relying on any information from LiDAR or cameras. To reduce the time required to run these synthesized programs, we first filter the input tracks based on confidence. We compute each track's confidence score by summing its confidence values across all timestamps. For the classes `REGULAR_VEHICLE`, `PEDESTRIAN`, `BOLLARD`, `CONSTRUCTION_CONE`, and `CONSTRUCTION_BARREL`, we retain the top 200 tracks per class. For all other classes, we retain the top 100 tracks per category. We prompt the LLM as follows:

Please use the following functions to find instances of a referred object in an autonomous driving dataset. Be precise to the description, try to avoid returning false positives.
API Listing: {API Listing}
Categories: {AV2 Categories}
Define a single scenario for the description:{Natural Language Description}
Here is a list of examples: {Prediction Examples}. Only output code and comments as part of a Python block. Do not define any additional functions, or filepaths. Do not include imports. Assume the log_dir and output_dir variables are given. Wrap all code in one python block and do not provide alternatives. Output code even if the given functions are not expressive enough to find the scenario.

We include the API listing in Appendix A.10. We found this prompt consistently produces reasonable code across all tested LLMs. After running the synthesized programs, we post-process the output tracks. We discard object relationships where the spatial distance exceeds 50 meters. For the remaining tracks, we apply timestamp dilation to reduce flickering of the `referred` object labels. We symmetrically dilate time segments to a minimum of 1.5 seconds. For example, if the base tracker tracks an object with ID 1 from 3.0 to 15.0 seconds at 2 Hz, and the synthesized code marks it as referred at timestamps 4.5 and 5.0, we update the final output to label the track as referred from 4.0 to 5.5 seconds. Finally, we downsample all output tracks to 2 Hz for evaluation.

A.2 Token Statistics for LLM Baselines

ReferGPT, *LLM APIs as a Black Box*, and *RefProg* can be viewed as different ways of providing scaffolding around an LLM for RMOT and scenario mining tasks. Most commercially available LLM APIs bill according to four cost categories: uncached input tokens, cached input tokens, output tokens, and code interpreter sessions (CIS). Input token count scales approximately linearly with the length of the input text, the number of input images, and seconds of input video. *ReferGPT* requires orders of magnitude more input and output tokens because it calls a VLM call for every tracked object at every timestamp. It also requires processing images, which can use more than 1000 tokens (depending on resolution) using GPT-5-mini. *LLM APIs as a Black Box* only makes as many calls as are there logs of data. However, inspecting the reasoning traces reveals that GPT-5 automatically went through multiple rounds of generating data processing code and verifying the output. The LLM also often re-generates the same processing code between logs. Initializing code interpreter sessions represents approximately half of the total cost ($38.49) of inference. RefProg makes as many

calls as there are unique scenario descriptions. Since the entire atomic function API must be given in context, it requires a relatively high number of input tokens. However, most of these tokens are cached between requests. The output of the RefProg is relatively compact because the LLM has access to high-level composable functions and does not need to write extensive Python code.

Table A.1. **Token Efficiency by LLM Baseline** RefProg achieves the highest scenario mining performance for the lowest cost. Cost is in USD for the OpenAI Platform in November 2025. We report the total number of tokens used millions. CIS refers to code interpreter sessions.

Method	LLM	HOTA-Temporal ↑	Cost ↓	Uncached Input	Cached Input	Output	CIS
ReferGPT	GPT-5-mini	20.7	150.64	372.9M	4.0M	28.7M	0
LLM API as Black Box	GPT-5	37.2	77.59	4.5M	0	3.4M	1283
RefProg (Ours)	GPT-5	42.3	13.89	0.6M	2.6M	1.3M	0

A.3 Comparison of Recent Datasets

We provide a qualitative comparison of language expressions found in RefAV with other popular RMOT-for-driving and VQA-for-driving datasets. In particular, we examine nuPrompt, Refer-KITTI, and nuScenes-QA. To obtain a representative sample of each dataset, we compute text embeddings for each expression using Qwen-3-Embedding-8B. We display the first 30 samples using farthest point sampling initialized from the two samples furthest apart from each other. Many expressions in nuPrompt deal with object attributes that are minimally relevant to driving such as car color, a person's gender, or a person's clothing. Refer-KITTI similarly only refers to annotated vehicle and pedestrian classes. nuScenes-QA asks questions that are relevant to driving, but the dataset is limited by its lack of human annotations and unnatural sounding questions. We present examples from each dataset below.

RefAV (Ours)

```
- the vehicle two cars behind the ego vehicle
- pedestrian walking with dog on sidewalk
- stationary object
- motorcycle at stop sign
- group of people
- construction workers working next to the road
- fire truck in a non-emergency
- excavator
- large truck blocking view of vehicle that is about to turn left onto a main road
```

- bicyclist changing lanes to the right
- car with a stroller to its left
- wheelchair user at pedestrian crossing
- person directing traffic in a school zone
- skateboarder on the road
- unattended dog
- accelerating wheeled devices
- construction barrel with at least 2 construction cones within 3 meters
- bus with multiple pedestrians waiting on the right side
- vehicle passing the ego vehicle going signifcantly over the speed limit on an urban road
- van facing the wrong way in the middle turn lane
- two of the same work truck
- pedestrian near message board trailer
- pedestrians over 30 meters away
- the vehicle behind another vehicle being crossed by a jaywalking pedestrian
- vehicle flashing their hazard lights
- ego vehicle driving while it is cloudy
- motorcycle or pedestrian within 5 meters to the right of the ego vehicle
- bicycle behind a bus
- scooter lying down on the side of the road
- vehicles being passed by a motorcycle on either side

nuPrompt [14]

- Back, the pedestrian
- a stationary orange parking is on the parking space
- at the parking space, the trailer
- Car waiting with the same direction
- A person on foot by the roadside
- The silver lightcolored parked car is stationary in the parking space
- The objects possess the color blue
- Males with a black bag are getting ready to cross the road
- The one riding the black lefthandturn bicycle is a person
- Men stand on the street wearing pants and hats
- Females walking across the street
- Men wear pants while walking across the street
- In a state of sitting

- The car is in the act of turning
- Pedestrian getting ready to cross over
- is in the process of making a turn
- The car in motion is black and moving in the opposite direction
- Males wear tshirts and pants while making a left turn on the opposite side of the road
- the pedestrian is situated at the back left
- The dark car is at thestandstill and awaiting with the opposite direction as the truck
- The truck is thepale silver color
- we see the car ahead
- Females walk while wearing pants in the same direction
- Security camera
- Men are wearing shorts while standing on the road
- There is the truck positioned in the parking spot
- Color is red
- The bus is in front
- On the pedestrian pathway
- The car that is moving in the same direction is blue

Refer-KITTI [13]

- cars which are faster than ours
- walking pedestrian in the left
- women back to the camera
- males in the right
- left cars which are parking
- people in the pants
- people in the pants
- people
- women carrying a bag
- light-color vehicles
- vehicles in the counter direction
- walking males
- standing females
- men in the left
- right cars in silver
- red moving cars

- women
- walking people in the right
- vehicles
- pedestrian
- vehicles in front of ours
- vehicles which are in the left and turning
- vehicles in white
- cars in horizon direction
- black vehicles in right
- standing men
- vehicles which are braking
- walking women
- men
- right cars in the same direction of ours
- black cars

nuScenes-QA [15]

- How many without rider motorcycles are there?
- There is a thing that is both to the front of me and the front left of the traffic cone; what is it?
- Does the trailer have the same status as the truck that is to the back of the trailer?
- How many pedestrians are to the back of the moving bus?
- What is the with rider thing that is to the back right of the with rider bicycle?
- Are there any things?
- Is there another barrier of the same status as the barrier?
- The car to the back right of the stopped construction vehicle is in what status?
- There is a parked truck; what number of barriers are to the front left of it?
- Are there any other bicycles that in the same status as the car?
- What status is the motorcycle that is to the front left of the standing pedestrian?
- The without rider thing that is to the front of the construction vehicle is what?
- What number of traffic cones are there?
- Are there any trailers to the front of me?
- What is the thing to the back of the stopped thing?
- There is a without rider thing; are there any parked cars to the back of it?
- What number of barriers are to the back of the motorcycle?

```
- Are there any animals to the back right of me?
- The thing that is to the back right of the truck and the back right of the bus is
    what?
- There is a bus; is its status the same as the thing that is to the front of the
   bus?
- There is a with rider bicycle; what number of moving things are to the front of
   it?
- Are there any moving construction vehicles?
- The standing pedestrian to the back right of the traffic cone is what?
- How many buss are there?
- Is the status of the truck to the back of the pedestrian the same as the truck to
    the front of the bicycle?
- Is there another thing that has the same status as the animal?
- Are there any things to the back of the moving trailer?
- Are there any moving buss to the front of the without rider thing?
- The thing that is to the front of the moving truck and the front left of the
   stopped trailer is what?
- How many stopped trailers are there?
```

A.4 Scenario Mining Annotation Tool

In order to manually annotate interesting scenarios, we build a custom web app with Claude 3.7 (Figure **Figure A.1**). To construct a scenario, the user selects an object by clicking on a point within the image frame, writes a natural language description, and selects the start and end frames of the `referred objects` correspond to the prompt. We project a ray originating from the 2D point and find the ground truth 3D bounding box centroid closest to the ray. This tool allows us to quickly generate multi-object and multi-camera referential tracks.

A.5 Referential Tracking Evaluation of `Related` and `Other` Objects

RefProg creates a scene graph for each natural language prompt that not only identifies `referred objects`, but also classifies tracks as `related objects` and `other objects`. For example, given the prompt `vehicle near a pedestrian`, the vehicle would be the `referred object`, the pedestrian would be the `related object`, and everything else in the scene would be considered `other objects`. We benchmark the base tracker with HOTA and evaluate HOTA-Temporal of `related objects` and `other objects` in Table **Table A.2**. Since most

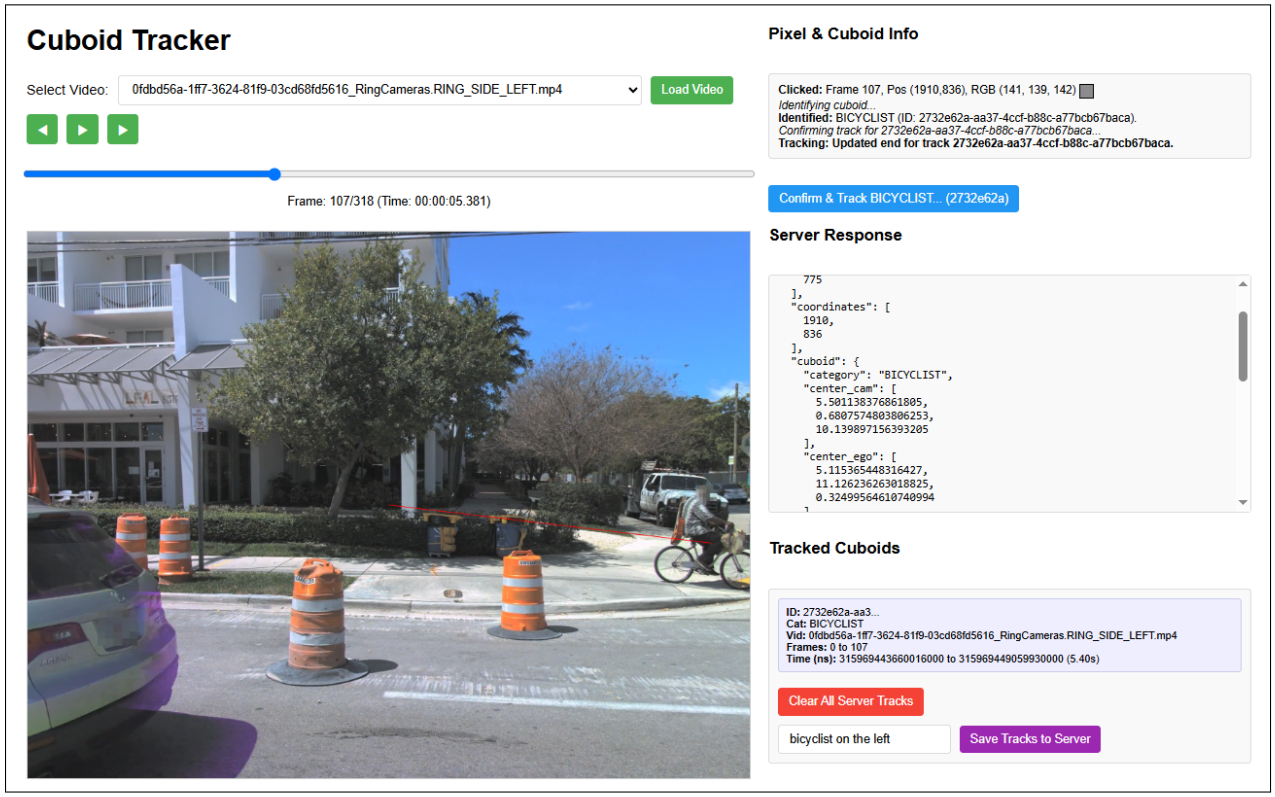


Figure A.1. **Manual Annotation Tool**. We create an annotation tool to assist with labeling manually defined scenarios. Our tool allows us to quickly annotate multi-object referential tracks in AV2.

objects are not related to the prompt, HOTA and HOTA-Temporal `Other` are similar.

Table A.2. **Referential Tracking Accuracy for `Related` and `Other` Objects.** We present RefProg's referential tracking accuracy for `related objects` and `other objects`. We find that HOTA (for the base tracker) is similar to HOTA-Temporal (`Other`) because most objects are not relevant to the referential prompt.

Method	HOTA ↑	HOTA-Temporal (`Related`) ↑	HOTA-Temporal (`Other`) ↑
Referential Tracking by Program Synthesis			
Ground Truth	100.0	57.2	97.9
Le3DE2E [72]	78.9	35.6	74.2
ReVoxelDet [73]	69.4	28.2	63.4
TransFusion [74]	69.5	28.6	63.8
Valeo4Cast [75]	75.1	30.8	69.2

A.6 Impact of Sampling Rate on RefProg

We evaluate the impact of sampling rate on RefProg. First, we subsample three detectors (Le3DE2E [72], Valeo4Cast [75], and BEVFusion [103]) at 2Hz and 10 Hz. Next, we associate all detections with AB3DMOT [104] to generate 2Hz and 10Hz tracks respectively. Lastly, we evaluate RefProg on all trackers at 2Hz. We find that standard tracking performance drops by 5% when subsampling due to the difficulty of associating sparse detections. We see a similar drop in HOTA-Track, Log Balanced Accuracy and Timestamp Balanced Accuracy. Interestingly, we do not see a similar drop in performance with HOTA-Temporal. This suggests that RefProg might be robust to temporally sparse inputs and ID switches.

Table A.3. **Impact of Sampling Rate.** We evaluate RefProg's performance with subsampled inputs to measure the relative impact on referential tracking accuracy. We find that standard tracking performance drops by 5% when subsampling due to the difficulty of associating sparse detections.

Method	Hz	HOTA ↑	HOTA-Temporal ↑	HOTA-Track ↑	Log Bal. Acc. ↑	Timestamp Bal. Acc. ↑
Ground Truth	10	100.0	64.7	68.7	81.1	80.7
Ground Truth	2	100.0	64.2	68.8	80.6	80.1
Le3DE2E Det. [72] + AB3DMOT [104]	10	78.0	48.3	49.2	73.1	73.6
Le3DE2E Det. [72] + AB3DMOT [104]	2	73.0	45.2	44.1	69.3	69.5
Valeo4Cast Det. [75] + AB3DMOT [104]	10	74.8	42.8	45.5	71.6	71.8
Valeo4Cast Det. [103] + AB3DMOT [104]	2	69.9	45.2	44.1	69.3	69.5
BEVFusion Det. [103] + AB3DMOT [104]	10	76.0	46.4	47.8	73.7	72.3
BEVFusion Det. [103] + AB3DMOT [104]	2	70.9	46.4	45.2	72.1	72.9

A.7 More Dataset Statistics

Figure Figure A.3 quantifies important characteristics of RefAV such as the number of unique scenario descriptions, the number of positive and negative examples, the number of

procedurally and manually defined scenarios, the number of scenarios that include object relationships, and the number of scenarios directly involving the ego-vehicle. We use the catch-all term "expression" for scenario description to better compare RefAV to other datasets. A positive match occurs when the data log contains at least one `referred object` corresponding to the scenario description. Figure Figure A.2 shows the number of positive and negative examples for representative scenario descriptions. In Figure Figure A.4, we evaluate the spatial distribution of `referred objects`.

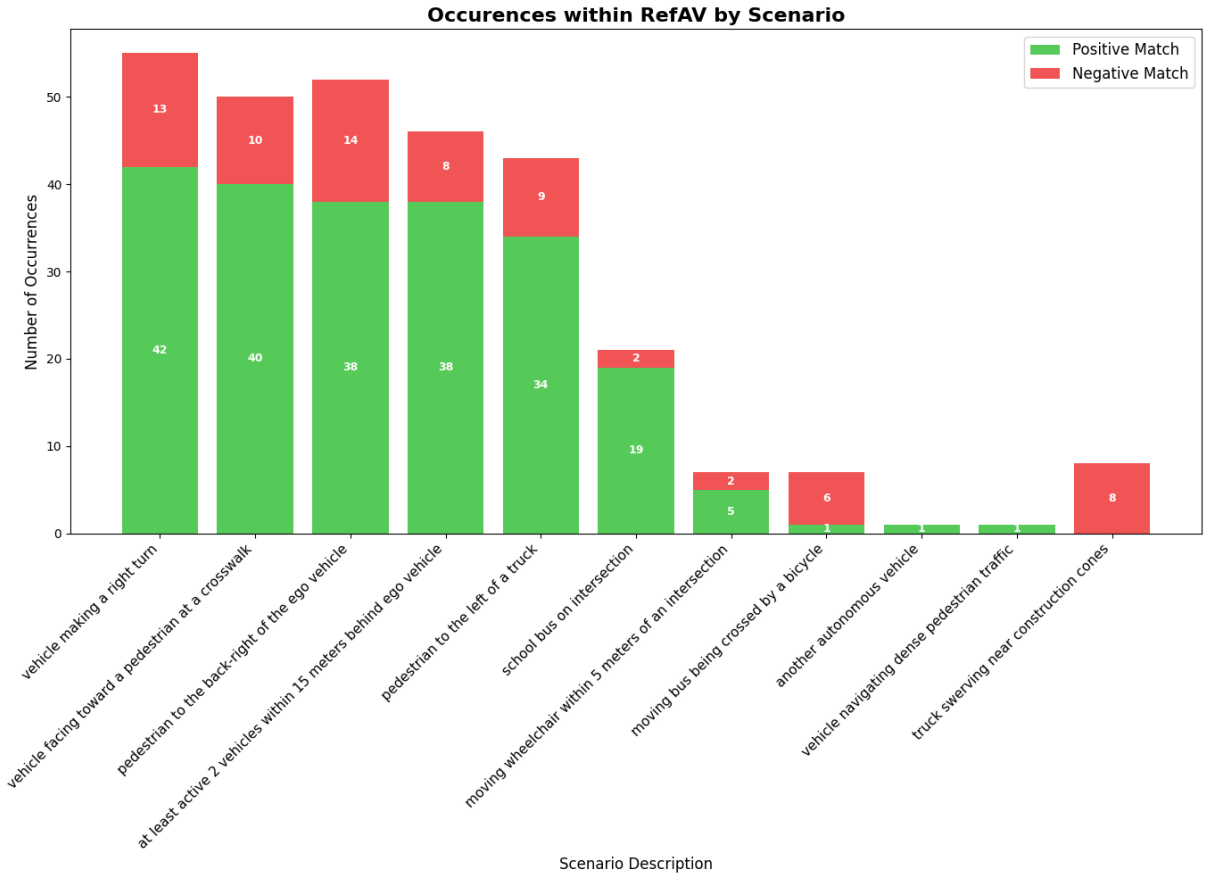


Figure A.2. **Occurrences within RefAV by Scenario.** RefAV contains 407 unique scenario descriptions. We display 11 representative descriptions picked uniformly. After the diversity sampling process, RefAV includes about 40 positive matches for common scenarios such as `vehicle making a right turn`. Less common scenarios such as `moving bus being crossed by a bicycle` occur only in only a handful of logs across the ArgoVerse 2 dataset. RefAV includes negative examples for almost all procedurally defined scenarios. Manually annotated scenarios such as `another autonomous vehicle` do not have corresponding negative examples.

Expression Type	# Expressions
Unique	407
Positive Match	7910
Negative Match	2090
Procedurally Defined	9880
Manually Defined	120
Includes `Related Objects`	5531
Ego is the only `Referred Object`	827
Ego is the only `Related Object`	1200

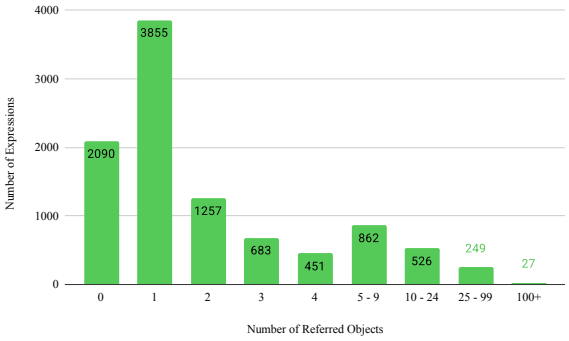


Figure A.3. **RefAV Expression Statistics.** We compute the number of referring expressions categorized by expression type, count the number of the expressions that are positive or negative, and whether they are procedurally generated or manually constructed. We also highlight the number of positive expressions that include `related object` annotations and explicit references to the ego vehicle. The bar chart on the right shows the number of objects refered to by each expression. Most expressions in RefAV refer to zero (e.g. a negative match) or one object. On average, each expression refers to 3.92 objects.

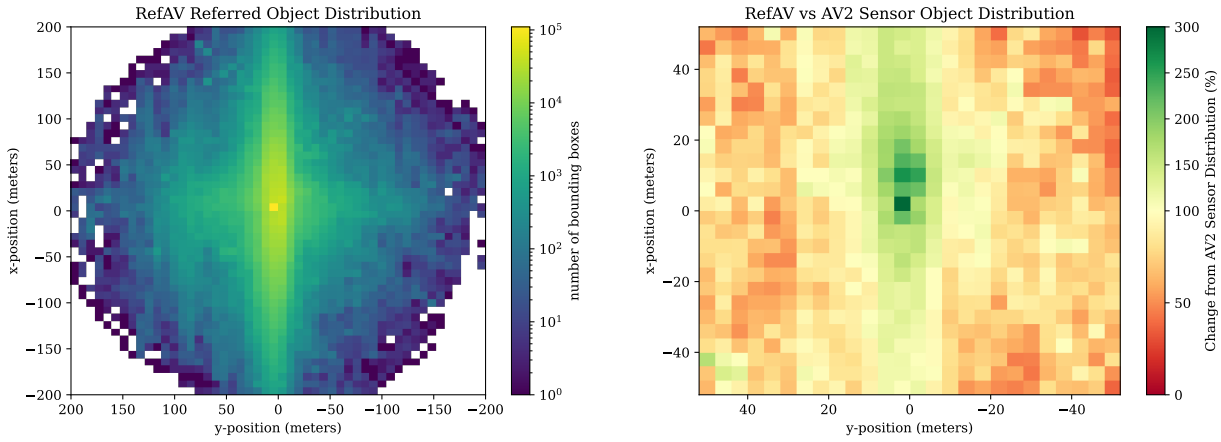


Figure A.4. **RefAV Object Distribution.** RefAV includes `referred objects` in all directions up to 150m away from the ego vehicle. RefAV places a special focus on the ego vehicle and objects that interact with the ego vehicle. Therefore, `referred objects` are disproportionately located on the road in front of the ego vehicle. Both `referred object` heatmaps are in the ego vehicle coordinate frame.

A.8 Summary of Competition Top Performers

We summarize the contributions of top teams below.

Team Zeekr UMCV extends RefProg with global context-aware generation to processes all queries collectively for improved reasoning and consistency. They also propose a multi-agent refinement system where a secondary refiner agent iteratively debugs and enhances the generated code.

Team Mi3 UCM proposes two key improvements to RefProg: (1) Fault-Tolerant Iterative Code Generation (FT-ICG), which refines faulty code by re-prompting the LLM with error feedback until successful execution, and (2) Enhanced Prompting for Spatial Relational Functions (EP-SRF), which clarifies function semantics to prevent misinterpretations of spatial relationships between objects.

Team ZXH refines RefProg by prompting LLMs with a batch of queries and in-context examples instead of prompting for each query one at a time. Importantly, this team's approach relies on in-context learning and batch prompting to improve consistency and efficiency.

A.9 Examples of RefProg's Synthesized Programs

We provide examples of RefProg's correctly synthesized programs below.

Prompt: Pedestrian crossing between stopped buses

```
# Description: pedestrian crossing between stopped buses

# First, let's get all pedestrians and buses from the dataset
pedestrians = get_objects_of_category(log_dir, category="PEDESTRIAN")
buses = get_objects_of_category(log_dir,  category="BUS")

# We need to find stopped buses
stationary_buses = stationary(buses, log_dir)

# Alternatively, we could use has_velocity with  a low threshold to find buses that are  temporarily stopped
stopped_buses = has_velocity(buses, log_dir, min_velocity=0, max_velocity=0.5)

# Now we need to find pedestrians that are crossing between the stopped buses. First, identify pedestrians that are crossing in front of a stopped bus
peds_crossing_bus = being_crossed_by(stationary_buses, pedestrians, log_dir)

```

```
# Reverse the relationship to get the pedestrians that are doing the crossing
crossing_peds = reverse_relationship(being_crossed_by)(stationary_buses, pedestrians, log_dir)

# Now we need to find pedestrians that have buses on both sides (left and right). This means the pedestrian is between buses
peds_with_bus_left = has_objects_in_relative_direction(crossing_peds, stationary_buses, log_dir, direction="left", min_number=1, within_distance=10)
peds_with_bus_right = has_objects_in_relative_direction(crossing_peds, stationary_buses, log_dir, direction="right", min_number=1, within_distance=10)

# Combine to find pedestrians with buses on both sides
peds_between_buses = scenario_and([peds_with_bus_left, peds_with_bus_right])

# Output the scenario
output_scenario(peds_between_buses, "pedestrian crossing between stopped buses", log_dir, output_dir)
```

Prompt: Group of at least 3 moving bicyclists with 5m from each other

```
# Description: Group of at least 3 moving bicyclists within 5 meters of each other

# Get all bicyclists from the dataset
bicyclists = get_objects_of_category(log_dir, category="BICYCLIST")

# Filter out stationary bicyclists to get only moving ones
moving_bicyclists = scenario_not(stationary)(bicyclists, log_dir)

# Find bicyclists that have at least 2 other bicyclists within 5 meters (which makes a group of at least 3 bicyclists)
bicyclists_in_groups = near_objects(moving_bicyclists, moving_bicyclists, log_dir, distance_thresh=5, min_objects=2, include_self=False)

# Output the scenario
output_scenario(bicyclists_in_groups, "group of at least 3 moving bicyclists within 5 meters of each other", log_dir, output_dir)
```

Prompt: Accelerating vehicle changing lanes to the right

```
# Description: accelerating vehicle changing lanes to the right
```

```
# First, get all vehicles in the scene
vehicles = get_objects_of_category(log_dir, category="VEHICLE")

# Find vehicles that are accelerating using the accelerating function with a reliable minimum acceleration threshold of 0.65
accelerating_vehicles = accelerating(vehicles, log_dir, min_accel=0.65)

# Find vehicles that are changing lanes to the right. We use the changing_lanes function with direction='right'
right_lane_changes = changing_lanes(vehicles, log_dir, direction='right')

# Now find the intersection of accelerating vehicles and those changing lanes to the right. This gives us vehicles that are both accelerating and changing lanes to the right
accelerating_and_changing_right = scenario_and([accelerating_vehicles, right_lane_changes])

# Output the scenario
output_scenario(accelerating_and_changing_right,
"accelerating vehicle changing lanes to the right",
log_dir, output_dir)
```

We provide examples of RefProg's incorrectly synthesized programs below.

Prompt: bus in front of ego vehicle obstructing view of red traffic light

The following program fails because the pre-defined atomic functions lack the expressivity to synthesize a correct program. Interestingly, the LLM realizes that the provided functions don't allow us to determine if a traffic light is being obstructed.

```
# Description: bus in front of ego vehicle obstructing view of red traffic light

# Get the ego vehicle and buses
ego_vehicle = get_objects_of_category(log_dir, category="EGO_VEHICLE")
buses = get_objects_of_category(log_dir, category="BUS")

# Find buses that are in front of the ego vehicle
# We need to find buses that are directly in front of the ego vehicle within a reasonable distance (20 meters) and with minimal lateral offset
buses_in_front_of_ego = get_objects_in_relative_direction(ego_vehicle, buses, log_dir, direction='forward', min_number=1, within_distance=20, lateral_thresh=3)

```

```
# We want buses that are moving in the same direction as the ego vehicle to ensure they're actually in the same lane and potentially blocking the view
buses_in_same_direction = heading_in_relative_direction_to(buses_in_front_of_ego, ego_vehicle, log_dir, direction='same')

# We need buses that are in the same lane as the ego vehicle
buses_in_same_lane = in_same_lane(buses_in_same_direction, ego_vehicle, log_dir)

# These buses are likely to be obstructing the view of traffic lights
# Note: The functions don't directly allow us to determine if a traffic light is being obstructed, but buses directly in front of the ego vehicle in the same lane are the most likely candidates for obstructing the view of traffic lights

output_scenario(buses_in_same_lane, "bus in front of ego vehicle obstructing view of red traffic light", log_dir, output_dir)
```

Prompt: bicycle following closely behind a vehicle

The following program fails due to imprecise semantics because the bicycle is not necessarily following the vehicle.

```
# Description: bicycle following closely behind a vehicle
# We need to find bicycles that are following vehicles closely

# Get all bicycles and vehicles from the dataset
bicycles = get_objects_of_category(log_dir, category="BICYCLE")
vehicles = get_objects_of_category(log_dir, category="VEHICLE")

# We need to identify bicycles that are behind vehicles using get_objects_in_relative_direction to find bicycles that are behind vehicles. The bicycles should be within a reasonable distance (e.g., 10 meters) to be considered "following closely"
bicycles_behind_vehicles = get_objects_in_relative_direction(vehicles, bicycles, log_dir, direction="backward", within_distance=10, lateral_thresh=2)

# We might also want to ensure the bicycle is moving (not stationary)
moving_bicycles = scenario_not(stationary)(bicycles, log_dir)

# Combine the conditions: bicycles that are both moving and following closely behind vehicles
```

```
bicycles_following_vehicles = scenario_and([bicycles_behind_vehicles, moving_bicycles])

# Output the scenario
output_scenario(bicycles_following_vehicles, "bicycle following closely behind a vehicle", log_dir, output_dir)
```

Prompt: vehicle starting to switch lanes but then going back

The following program uses incorrect logic. It fails to combine lane changes and acceleration and uses AND on exclusive sets.

```
# Strategy:
# 1. Get all vehicles
# 2. Find vehicles that are changing lanes
# 3. Find vehicles that are also changing lanes in the opposite direction shortly after
# 4. This indicates a vehicle that started to change lanes but then went back

# Get all vehicles
vehicles = get_objects_of_category(log_dir, category="VEHICLE")

# Find vehicles that are moving (not stationary)
moving_vehicles = scenario_not(stationary)(vehicles, log_dir)

# Find vehicles that are changing lanes in either direction
lane_changing_vehicles = changing_lanes(moving_vehicles, log_dir)

# Find vehicles that are changing lanes to the left
left_lane_changes = changing_lanes(moving_vehicles, log_dir, direction='left')

# Find vehicles that are changing lanes to the right
right_lane_changes = changing_lanes(moving_vehicles, log_dir, direction='right')

# A vehicle that starts to switch lanes but goes back would show lateral acceleration in one direction followed by lateral acceleration in the opposite direction. We can look for vehicles with significant lateral acceleration in both directions

# Find vehicles with lateral acceleration to the left
left_accel = has_lateral_acceleration(moving_vehicles, log_dir, min_accel=0.5)
```

```
A.26
A.27 # Find vehicles with lateral acceleration to the right
A.28 right_accel = has_lateral_acceleration(moving_vehicles, log_dir, min_accel
        =-0.5, max_accel=0)
A.29
A.30 # Vehicles that have both left and right acceleration patterns are likely
        the ones that started to change lanes but went back
A.31 lane_change_aborted = scenario_and([left_accel, right_accel])
A.32 # Output the scenario
A.33 output_scenario(lane_change_aborted, description, log_dir, output_dir)
```

A.10 API Listing

We present our full Python API listing below.

```
A.1  def has_objects_in_relative_direction(
A.2      track_candidates:dict,
A.3      related_candidates:dict,
A.4      log_dir:Path,
A.5      direction:Literal["forward", "backward", "left", "right"],
A.6      min_number:int=1,
A.7      max_number:int=np.inf,
A.8      within_distance:float=50,
A.9      lateral_thresh:float=np.inf)-> dict:
A.10     """
A.11     Identifies tracked objects with at least the minimum number of related
         candidates in the specified direction. If the minimum number is met,
        will create relationships equal to the max_number of closest objects.
A.12
A.13     Args:
A.14         track_candidates: Tracks to analyze (scenario dictionary).
        related_candidates: Candidates to check for in direction (scenario
        dictionary).
A.15         log_dir: Path to scenario logs.
A.16         direction: Direction to analyze from the track's point of view ('
        forward', 'backward', 'left', 'right').
A.17         min_number: Minimum number of objects to identify in the direction
         per timestamp. Defaults to 1.
A.18         max_number: Maximum number of objects to identify in the direction
         per timestamp. Defaults to infinity.
A.19         within_distance: Maximum distance for considering an object in the
         direction. Defaults to infinity.
```

```
        lateral_thresh: Maximum lateral distance the related object can be from the sides of the tracked object. Defaults to infinity.

    Returns:
        dict: A scenario dictionary where keys are track UUIDs and values are dictionaries containing related candidate UUIDs and lists of timestamps when the condition is met for that relative direction.

    Example:
        vehicles_with_peds_in_front = has_objects_in_relative_direction(vehicles, pedestrians, log_dir, direction='forward', min_number=2)
    """


def get_objects_in_relative_direction(
    track_candidates:dict,
    related_candidates:dict,
    log_dir:Path,
    direction:Literal["forward", "backward", "left", "right"],
    min_number:int=0,
    max_number:int=np.inf,
    within_distance:float=50,
    lateral_thresh:float=np.inf)->dict:
    """
    Returns a scenario dictionary of the related candidates that are in the relative direction of the track candidates.


    Args:
        track_candidates: Tracks (scenario dictionary).
        related_candidates: Candidates to check for in direction (scenario dictionary).
        log_dir: Path to scenario logs.
        direction: Direction to analyze from the track's point of view ('forward', 'backward', 'left', 'right').
        min_number: Minimum number of objects to identify in the direction per timestamp. Defaults to 0.
        max_number: Maximum number of objects to identify in the direction per timestamp. Defaults to infinity.
        within_distance: Maximum distance for considering an object in the direction. Defaults to infinity.
        lateral_thresh: Maximum lateral distance the related object can be from the sides of the tracked object. Lateral distance is distance is
```

```
    the distance from the sides of the object that are parallel to the
    specified direction. Defaults to infinity.
A.52
A.53     Returns:
A.54         dict: A scenario dictionary where keys are track UUIDs and values
    are dictionaries containing related candidate UUIDs and lists of
    timestamps when the condition is met for that relative direction.
A.55
A.56     Example:
A.57         peds_in_front_of_vehicles = get_objects_in_relative_direction(
    vehicles, pedestrians, log_dir, direction='forward', min_number=2)
A.58     """
A.59
A.60
A.61 def get_objects_of_category(
A.62     log_dir,
A.63     category)->dict:
A.64     """
A.65     Returns all objects from a given category from the log annotations.
    This method accepts the super-categories "ANY" and "VEHICLE".
A.66
A.67     Args:
A.68         log_dir: Path to the directory containing scenario logs and data.
A.69         category: the category of objects to return
A.70
A.71     Returns:
A.72         dict: A scenario dict that where keys are the unique id (uuid) of
    the object and values are the list of timestamps the object is in view
    of the ego-vehicle.
A.73
A.74     Example:
A.75         trucks = get_objects_of_category(log_dir, category='TRUCK')
A.76     """
A.77
A.78
A.79 def is_category(
A.80     track_candidates:dict,
A.81     log_dir:Path,
A.82     category:str)->dict:
A.83     """
A.84     Returns all objects from a given category from track_candidates dict.
    This method accepts the super-categories "ANY" and "VEHICLE".
A.85
```

```
    Args:
        track_candidates: The scenario dict containing the objects to filter down
        log_dir: Path to the directory containing scenario logs and data.
        category: the category of objects to return

    Returns:
        dict: A scenario dict that where keys are the unique id of the object of the given category and values are the list of timestamps the object is in view of the ego-vehicle.

    Example:
        box_trucks = is_category(vehicles, log_dir, category='BOX_TRUCK')
    """


def turning(
    track_candidates: dict,
    log_dir:Path,
    direction:Literal["left", "right", None]=None)->dict:
    """
    Returns objects that are turning in the given direction.

    Args:
        track_candidates: The objects you want to filter from (scenario dictionary).
        log_dir: Path to scenario logs.
        direction: The direction of the turn, from the track's point of view ('left', 'right', None).

    Returns:
        dict: A filtered scenario dictionary where keys are track UUIDs that meet the turning criteria and values are nested dictionaries containing timestamps.

    Example:
        turning_left = turning(vehicles, log_dir, direction='left')
    """


def changing_lanes(
    track_candidates:dict,
    log_dir:Path,
```

```
    direction:Literal["left", "right", None]=None)-> dict:
    """
    Identifies lane change events for tracked objects in a scenario.

    Args:
        track_candidates: The tracks to analyze (scenario dictionary).
        log_dir: Path to scenario logs.
        direction: The direction of the lane change. None indicates
   tracking either left or right lane changes ('left', 'right', None).

    Returns:
        dict: A filtered scenario dictionary where keys are track UUIDs
   that meet the lane change criteria and values are nested dictionaries
   containing timestamps and related data.

    Example:
        left_lane_changes = changing_lanes(vehicles, log_dir, direction='
   left')
    """


def has_lateral_acceleration(
    track_candidates:dict,
    log_dir:Path,
    min_accel=-np.inf,
    max_accel=np.inf)-> dict:
    """
    Objects with a lateral acceleartion between the minimum and maximum
   thresholds. Most objects with a high lateral acceleration are turning.
   Postive values indicate accelaration to the left while negative values
   indicate acceleration to the right.

    Args:
        track_candidates: The tracks to analyze (scenario dictionary).
        log_dir: Path to scenario logs.
        direction: The direction of the lane change. None indicates
   tracking either left or right lane changes ('left', 'right', None).

    Returns:
        dict: A filtered scenario dictionary where keys are track UUIDs
   that meet the lane change criteria and values are nested dictionaries
   containing timestamps and related data.

```

```
    Example:
        jerking_left = has_lateral_acceleration(non_turning_vehicles,
    log_dir, min_accel=2)
    """


def facing_toward(
    track_candidates:dict,
    related_candidates:dict,
    log_dir:Path,
    within_angle:float=22.5,
    max_distance:float=50)->dict:
    """
    Identifies objects in track_candidates that are facing toward objects
    in related candidates. The related candidate must lie within a region
    lying within within_angle degrees on either side the track-candidate's
    forward axis.

    Args:
        track_candidates: The tracks that could be heading toward another
    tracks
        related_candidates: The objects to analyze to see if the
    track_candidates are heading toward
        log_dir:  Path to the directory containing scenario logs and data.
        fov: The field of view of the track_candidates. The related
    candidate must lie within a region lying within fov/2 degrees on either
     side the track-candidate's forward axis.
        max_distance: The maximum distance a related_candidate can be away
     to be considered by

    Returns:
        dict: A filtered scenario dict that contains the subset of track
    candidates heading toward at least one of the related candidates.

    Example:
        pedestrian_facing_away = scenario_not(facing_toward)(pedestrian,
    ego_vehicle, log_dir, within_angle=180)
    """


def heading_toward(
    track_candidates:dict,
    related_candidates:dict,
```

```
    log_dir:Path,
    angle_threshold:float=22.5,
    minimum_speed:float=.5,
    max_distance:float=np.inf)->dict:
    """
    Identifies objects in track_candidates that are heading toward objects in related candidates. The track candidates acceleartion vector must be within the given angle threshold of the relative position vector. The track candidates must have a component of velocity toward the related candidate greater than the minimum_accel.

    Args:
        track_candidates: The tracks that could be heading toward another tracks
        related_candidates: The objects to analyze to see if the track_candidates are heading toward
        log_dir:  Path to the directory containing scenario logs and data.
        angle_threshold: The maximum angular difference between the velocity vector and relative position vector between the track candidate and related candidate.
        min_vel: The minimum magnitude of the component of velocity toward the related candidate
        max_distance: Distance in meters the related candidates can be away from the track candidate to be considered

    Returns:
        dict: A filted scenario dict that contains the subset of track candidates heading toward at least one of the related candidates.

    Example:
        heading_toward_traffic_cone = heading_toward(vehicles, traffic_cone, log_dir)
    """


def accelerating(
    track_candidates:dict,
    log_dir:Path,
    min_accel:float=.65,
    max_accel:float=np.inf)->dict:
    """
    Identifies objects in track_candidates that have a forward acceleration above a threshold. Values under -1 reliably indicates
```

```
    braking. Values over 1.0 reliably indiciates accelerating.
A.217
A.218     Args:
A.219         track_candidates: The tracks to analyze for acceleration (scenario
          dictionary)
A.220         log_dir:  Path to the directory containing scenario logs and data.
A.221         min_accel: The lower bound of acceleration considered
A.222         max_accel: The upper bound of acceleration considered
A.223
A.224     Returns:
A.225         dict: A filtered scenario dictionary containing the objects with
          an acceleration between the lower and upper bounds.
A.226
A.227     Example:
A.228         accelerating_motorcycles = accelerating(motorcycles, log_dir)
A.229     """
A.230
A.231
A.232 def has_velocity(
A.233     track_candidates:dict,
A.234     log_dir:Path,
A.235     min_velocity:float=.5,
A.236     max_velocity:float=np.inf)->dict:
A.237     """
A.238     Identifies objects with a velocity between the given maximum and
          minimum velocities in m/s. Stationary objects may have a velocity up to
          0.5 m/s due to annotation jitter.
A.239
A.240     Args:
A.241         track_candidates: Tracks to analyze (scenario dictionary).
A.242         log_dir: Path to scenario logs.
A.243         min_velocity: Minimum velocity (m/s). Defaults to 0.5.
A.244         max_velocity: Maximum velocity (m/s)
A.245
A.246     Returns:
A.247         dict: Filtered scenario dictionary of objects meeting the velocity
          criteria.
A.248
A.249     Example:
A.250         fast_vehicles = has_min_velocity(vehicles, log_dir, min_velocity
          =5)
A.251     """
A.252
```

```
A.253
A.254 def at_pedestrian_crossing(
A.255     track_candidates:dict,
A.256     log_dir:Path,
A.257     within_distance:float=1)->dict:
A.258     """
A.259     Identifies objects that within a certain distance from a pedestrian
      crossing. A distance of zero indicates that the object is within the
      boundaries of the pedestrian crossing.
A.260
A.261     Args:
A.262         track_candidates: Tracks to analyze (scenario dictionary).
A.263         log_dir: Path to scenario logs.
A.264         within_distance: Distance in meters the track candidate must be
      from the pedestrian crossing. A distance of zero means that the object
      must be within the boundaries of the pedestrian crossing.
A.265
A.266     Returns:
A.267         dict: Filtered scenario dictionary where keys are track UUIDs and
      values are lists of timestamps.
A.268
A.269     Example:
A.270         vehicles_at_ped_crossing = at_pedestrian_crossing(vehicles,
      log_dir)
A.271     """
A.272
A.273
A.274 def on_lane_type(
A.275     track_uuid:dict,
A.276     log_dir,
A.277     lane_type:Literal["BUS", "VEHICLE", "BIKE"])->dict:
A.278     """
A.279     Identifies objects on a specific lane type.
A.280
A.281     Args:
A.282         track_candidates: Tracks to analyze (scenario dictionary).
A.283         log_dir: Path to scenario logs.
A.284         lane_type: Type of lane to check ('BUS', 'VEHICLE', or 'BIKE').
A.285
A.286     Returns:
A.287         dict: Filtered scenario dictionary where keys are track UUIDs and
      values are lists of timestamps.
A.288
```

```
    Example:
        vehicles_on_bus_lane = on_lane_type(vehicles, log_dir, lane_type="BUS")
    """


def near_intersection(
    track_uuid:dict,
    log_dir:Path,
    threshold:float=5)->dict:
    """
    Identifies objects within a specified threshold of an intersection in meters.

    Args:
        track_candidates: Tracks to analyze (scenario dictionary).
        log_dir: Path to scenario logs.
        threshold: Distance threshold (in meters) to define "near" an intersection.

    Returns:
        Filtered scenario dictionary where keys are track UUIDs and values are lists of timestamps.

    Example:
        bicycles_near_intersection = near_intersection(bicycles, log_dir, threshold=10.0)
    """


def on_intersection(
    track_candidates:dict,
    log_dir:Path)->dict:
    """
    Identifies objects located on top of an road intersection.

    Args:
        track_candidates: Tracks to analyze (scenario dictionary).
        log_dir: Path to scenario logs.

    Returns:
        dict: Filtered scenario dictionary where keys are track UUIDs and values are lists of timestamps.
```

```
A.326
A.327     Example:
A.328         strollers_on_intersection = on_intersection(strollers, log_dir)
A.329     """
A.330
A.331
A.332 def being_crossed_by(
A.333     track_candidates:dict,
A.334     related_candidates:dict,
A.335     log_dir:Path,
A.336     direction:Literal["forward", "backward", "left",
A.337     "right"]="forward",
A.338     in_direction:Literal['clockwise','counterclockwise',
A.339     'either']='either',
A.340     forward_thresh:float=10,
A.341     lateral_thresh:float=5)->dict:
A.342     """
A.343     Identifies objects that are being crossed by one of the related
          candidate objects. A crossing is defined as the related candidate's
          centroid crossing the half-midplane of a tracked candidate. The
          direction of the half-midplane is specified with the direction.
A.344
A.345     Args:
A.346         track_candidates: Tracks to analyze .
A.347         related_candidates: Candidates (e.g., pedestrians or vehicles) to
          check for crossings.
A.348         log_dir: Path to scenario logs.
A.349         direction: specifies the axis and direction the half midplane
          extends from
A.350         in_direction: which direction the related candidate has to cross
          the midplane for it to be considered a crossing
A.351         forward_thresh: how far the midplane extends from the edge of the
          tracked object
A.352         lateral_thresh: the two planes offset from the midplane. If an
          related candidate crosses the midplane, it will continue being
          considered crossing until it goes past the lateral_thresh.
A.353
A.354     Returns:
A.355         dict: A filtered scenario dictionary containing all of the track
          candidates that were crossed by the related candidates given the
          specified constraints.
A.356
A.357     Example:
```

```
        overtaking_on_left = being_crossed_by(moving_cars, moving_cars, log_dir, direction="left", in_direction="clockwise", forward_thresh=4)
        vehicles_crossed_by_peds = being_crossed_by(vehicles, pedestrians, log_dir)
    """


def near_objects(
    track_uuid:dict,
    candidate_uuids:dict,
    log_dir:Path,
    distance_thresh:float=10,
    min_objects:int=1,
    include_self:bool=False)->dict:
    """
    Identifies timestamps when a tracked object is near a specified set of related objects.

    Args:
        track_candidates: Tracks to analyze (scenario dictionary).
        related_candidates: Candidates to check for proximity (scenario dictionary).
        log_dir: Path to scenario logs.
        distance_thresh: Maximum distance in meters a related candidate can be away to be considered "near".
        min_objects: Minimum number of related objects required to be near the tracked object.

    Returns:
        dict: A scenario dictionary where keys are timestamps when the tracked object is near the required number of related objects and values are lists of related candidate UUIDs present at those timestamps.

    Example:
        vehicles_near_ped_group = near_objects(vehicles, pedestrians, log_dir, min_objects=3)
    """


def following(
    track_uuid:dict,
    candidate_uuids:dict,
```

```
    log_dir:Path)-> dict:
    """
    Returns timestamps when the tracked object is following a lead object.
    Following is defined simultaneously moving in the same direction and
    lane.
    """


def heading_in_relative_direction_to(
    track_candidates,
    related_candidates,
    log_dir,
    direction:Literal['same', 'opposite', 'perpendicular'])->dict:
    """
    Returns the subset of track candidates that are traveling in the given
    direction compared to the related candidates.

    Arguments:
        track_candidates: The set of objects that could be traveling in
    the given direction
        related_candidates: The set of objects that the direction is
    relative to
        log_dir: The path to the log data
        direction: The direction that the positive tracks are
        traveling in relative to the related candidates:
            - "opposite" indicates the track candidates are traveling in a
     direction 135-180 degrees from the direction the related candidates
    are heading toward.
            - "same" indicates the track candidates that are traveling in
    a direction 0-45 degrees from the direction the related candiates are
    heading toward.
            - "perpendicular" indicates the track candidates that are
    traveling in a direction 45-135 degrees from the direction the related
    candiates are heading toward.

    Returns:
        dict: the subset of track candidates that are traveling in the
    given direction compared to the related candidates.

    Example:
        oncoming_traffic = heading_in_relative_direction_to(vehicles,
    ego_vehicle, log_dir, direction='opposite')
    """
```

```
A.421
A.422
A.423 def stationary(
A.424     track_candidates:dict,
A.425     log_dir:Path)->dict:
A.426     """
A.427     Returns objects that moved less than 2m over their length of observation in the scneario. This object is only intended to separate parked from active vehicles. Use has_velocity() with thresholding if you want to indicate vehicles that are temporarily stopped.
A.428
A.429     Args:
A.430         track_candidates: Tracks to analyze (scenario dictionary).
A.431         log_dir: Path to scenario logs.
A.432
A.433     Returns:
A.434         dict: A filtered scenario dictionary where keys are track UUIDs and values are lists of timestamps when the object is stationary.
A.435
A.436     Example:
A.437         parked_vehicles = stationary(vehicles, log_dir)
A.438     """
A.439
A.440
A.441 def at_stop_sign(
A.442     track_candidates:dict,
A.443     log_dir:Path,
A.444     forward_thresh:float=10)->dict:
A.445     """
A.446     Identifies timestamps when a tracked object is in a lanee corresponding to a stop sign. The tracked object must be within 15m of the stop sign. This may highlight vehicles using street parking near a stopped sign.
A.447
A.448     Args:
A.449         track_candidates: Tracks to analyze (scenario dictionary).
A.450         log_dir: Path to scenario logs.
A.451         forward_thresh: Distance in meters the vehcile is from the stop sign in the stop sign's front direction
A.452
A.453     Returns:
A.454         dict: A filtered scenario dictionary where keys are track UUIDs and values are lists of timestamps when the object is at a stop sign.
```

```
A.455
A.456     Example:
A.457         vehicles_at_stop_sign = at_stop_sign(vehicles, log_dir)
A.458     """
A.459
A.460
A.461 def in_drivable_area(
A.462     track_candidates:dict,
A.463     log_dir:Path)->dict:
A.464     """
A.465     Identifies objects within track_candidates that are within a drivable
          area.
A.466
A.467     Args:
A.468         track_candidates: Tracks to analyze (scenario dictionary).
A.469         log_dir: Path to scenario logs.
A.470
A.471     Returns:
A.472         dict: A filtered scenario dictionary where keys are track UUIDs
          and values are lists of timestamps when the object is in a drivable
          area.
A.473
A.474     Example:
A.475         buses_in_drivable_area = in_drivable_area(buses, log_dir)
A.476     """
A.477
A.478
A.479 def on_road(
A.480     track_candidates:dict,
A.481     log_dir:Path)->dict:
A.482     """
A.483     Identifies objects that are on a road or bike lane. This function
          should be used in place of in_driveable_area() when referencing objects
           that are on a road. The road does not include  parking lots or other
          driveable areas connecting the road to parking lots.
A.484
A.485     Args:
A.486         track_candidates: Tracks to filter (scenario dictionary).
A.487         log_dir: Path to scenario logs.
A.488
A.489     Returns:
A.490         dict: The subset of the track candidates that are currently on a
          road.
```

```
    Example:
        animals_on_road = on_road(animals, log_dir)
    """


def in_same_lane(
    track_candidates:dict,
    related_candidates:dict,
    log_dir:Path)->dict:
    """
    Identifies tracks that are in the same road lane as a related candidate.

    Args:
        track_candidates: Tracks to filter (scenario dictionary)
        related_candidates: Potential objects that could be in the same lane as the track (scenario dictionary)
        log_dir: Path to scenario logs.

    Returns:
        dict: A filtered scenario dictionary where keys are track UUIDs and values are lists of timestamps when the object is on a road lane.

    Example:
        bicycle_in_same_lane_as_vehicle = in_same_lane(bicycle, regular_vehicle, log_dir)
    """

def on_relative_side_of_road(
    track_candidates:dict,
    related_candidates:dict,
    log_dir:Path,
    side=Literal['same', 'opposite'])->dict:
    """
    Identifies tracks that are in the same road lane as a related candidate.

    Args:
        track_candidates: Tracks to filter (scenario dictionary)
        related_candidates: Potential objects that could be in the same lane as the track (scenario dictionary)
        log_dir: Path to scenario logs.
```

```
    Returns:
        dict: A filtered scenario dictionary where keys are track UUIDs and values are lists of timestamps when the object is on a road lane.

    Example:
        bicycle_in_same_lane_as_vehicle = in_same_lane(bicycle, regular_vehicle, log_dir)
    """

def is_color(
    track_candidates: dict,
    log_dir: Path,
    color:Literal["white", "silver", "black", "red", "yellow", "blue"])->dict:
    """
    Returns objects that are the given color, determined by SIGLIP2 feature similarity.

    Args:
        track_candidates: The objects you want to filter from (scenario dictionary).
        log_dir: Path to scenario logs.
        color: The color of the objects you want to return. Must be one of 'white', 'silver', 'black', 'red', 'yellow', or 'blue'. Inputting a different color defaults to returning all objects.

    Returns:
        dict: A filtered scenario dictionary where keys are track UUIDs that meet the turning criteria and values are nested dictionaries containing timestamps.

    Example:
        red_cars = is_color(cars, log_dir, color='red')
    """

def scenario_and(scenario_dicts)->dict:
    """
    Returns a composed scenario where the track objects are the intersection of all of the track objects with the same uuid and timestamps.

    Args:
```

```
        scenario_dicts: the scenarios to combine

    Returns:
        dict: a filtered scenario dictionary that contains tracked objects found in all given scenario dictionaries

    Example:
        jaywalking_peds = scenario_and([peds_on_road, peds_not_on_pedestrian_crossing])
    """


def scenario_or(scenario_dicts)->dict:
    """
    Returns a composed scenario where that tracks all objects and relationships in all of the input scenario dicts.

    Args:
        scenario_dicts: the scenarios to combine

    Returns:
        dict: an expanded scenario dictionary that contains every tracked object in the given scenario dictionaries

    Example:
        be_cautious_around = scenario_or([animal_on_road, stroller_on_road])
    """


def reverse_relationship(func)->dict:
    """
    Wraps relational functions to switch the top level tracked objects and relationships formed by the function.

    Args:
        relational_func: Any function that takes track_candidates and related_candidates as its first and second arguements

    Returns:
        dict: scenario dict with swapped top-level tracks and related candidates

```

```
    Example:
        group_of_peds_near_vehicle = reverse_relationship(near_objects)(vehicles, peds, log_dir, min_objects=3)
    """


def scenario_not(func)->dict:
    """
    Wraps composable functions to return the difference of the input track dict and output scenario dict. Using scenario_not with a composable relational function will not return any relationships.

    Args:
        composable_func: Any function that takes track_candidates as its first input

    Returns:
        dict: scenario dict with tracks that are not filtered by func

    Example:
        active_vehicles = scenario_not(stationary)(vehicles, log_dir)
    """


def output_scenario(
    scenario:dict,
    description:str,
    log_dir:Path,
    output_dir:Path,
    visualize:bool=False,
    **visualization_kwargs):
    """
    Outputs a file containing the predictions in an evaluation-ready format. Do not provide any visualization kwargs.
    """
```

Appendix B
Supplemental Material for CAL

B.1 Language Annotation Details

We auto-label our training trajectories with three sets of language annotations. These are "naive Gemma", "naive Gemini", and "Gemini with CAL" experiments found in Table 3.2. The following sections detail the prompt given to the annotator VLM.

Naive Gemma Annotation Prompt

> Answer the following questions. Focus on the red route, what direction does it go or turn? Does it leave the trail? Are there roads or trails visible in the image? If yes, is the vehicle currently on one? Are there any nearby terrain features or objects that are relevant to the path the vehicle should take? Are there any far-range objects relevant to navigation? Is the camera degraded in any way?
> After answering these questions, please generate a command that would let the driver know to proceed down the red route. Prioritize far-range terrain features, but only if they are in the direction of the red route. Do not mention the drawn route. Ouput one command and wrap it in brackets.
> Simple example: [Go down the trail.] Complex example (assume you are driving in a rocky desert with two large rock formations in the distance): [Head toward the right rock formation as quickly as possible. After getting there, go around the left side.]

Naive Gemini Annotation Prompt

> The vehicle being driven is an ATV. Please generate a language instruction that would let a driver know to take the red route. If there is no red route, that indicates that the driver is not moving. Please use landmarks and/or terrain condition. If there is something notable in the image that the trajectory skirts around or diverges away from,

please you should include it in your instruction. Avoid using the words left or right unless it is absolutely necessary to differentiate between potential routes.
For your instruction, point to 1-2 objects within the image that correspond to that instruction. Do not mention the overlaid route. The points are in [y, x] format normalized to 0-1000.
Your output should be in JSON format

```
‘‘‘json
{
    "instruction": <instruction>,
    "image_points": [{"point": <point>, "label": <label1>}, ...]
}
‘‘‘
```

Gemini with CAL Annotation Prompt

The vehicle is approaching an intersection. Please generate two sets of instructions that would let a driver know to take the orange and cyan routes. Please use landmarks and/or terrain condition. You avoid using the words left or right unless it is absolutely necessary to differentiate between the routes.
For each instruction, point to the 1-2 objects within the image that correspond to that instruction. Do not mention the overlaid routes.The points are in [y, x] format normalized to 0-1000.
Your output should be in JSON format

```
‘‘‘json
{{
    "cyan_instructions": <instructions>,
    "cyan_image_points": [{"point": <point>, "label": <label1>}, ...],
    "orange_instructions": <instructions>,
    "orange_image_points": [{"point": <point>, "label": <label1>}, ...]
}}
‘‘‘
```

Test-Set Language

We test our models on human-labeled commands during open loop evaluation. We compute the language embeddings of all language commands using Qwen-3-Embedding-8B. We display a representative subset by taking the first 25 commands from farthest point sampling on the language embeddings.

```
- Go over that bump and get on the other trail.
- Execute a route that inspects all of the power line towers.
- Take a hard left.
- Avoid being seen in wide open areas. Get behind those bushes to avoid being seen.
- Park to the side right right up to the edge of the grass.
- Turn off the road we are currently on.
- Veer sharply towards the bright sun glare, cutting across the patch of open
    ground.
- Do a wide donut, keeping the fallen log and trash pile just to your right.
- Straighten out of this curve.
- Drive across the deep muddy ruts.
- Take a right turn. Hug the inside of the turn, driving across the shorter, mowed
    grass.
- Follow the tighter line that stays closer to the open meadow.
- Take the fork that goes to the yellow colored bushes
- Drive straight ahead along the main gravel track, keeping the tall utility pole
    in the distance as a guide.
- Pass by the white debris pile.
- Take a delayed turn, waiting for the lens flare to adjust so you can see where
    you are going.
- Take the most established path.
- Merge left onto the intersecting road.
- Go down the hill hugging the left side of the wildflowers.
- Pass through the field between the two two tall sets of weeds.
- I want to get to the top.
- Turn away from the sun.
- Drive between the log and trash pile.
- Turn off the main path into the wide, gravel-covered clearing, heading toward the
     dense tree line bordering the open area.
- Continue towards the house in the distance.
- There is a lot of water in the area. Take trails with no sitting water.
```

B.2 VLA Training Implementation

We train all models under the same hyperparameters using the Huggingface Transformers library. Each of the hyperparameters in Table B.1 was found to be reasonable across a range of models. Notably, we found little difference in the final performance of a model using LoRA vs full fine-tuning. Finally, we found that freezing the language embeddings of the base VLM to be important for language following.

Table B.1. **Training Hyperparameters.**

Hyperparameter	**Value**
Optimizer	AdamW
LoRA rank	128
LoRA alpha	16
Learning rate	5×10^{-5}
Vision encoder learning rate	1×10^{-5}
Batch size	48
Freeze vision	False
Freeze language embeddings	True
Weight decay	0.0
Max gradient norm	0.3
Num epochs	1
Warmup steps	50

We train our models using the below prompt template. The parts highlighted in blue correspond to the tokens that are updated according to the training sample's front camera image, past trajectory, and language command. The part highlighted in red corresponds to the agent's response. Loss is only calculated using the tokens that are part of the agent's response.

You are an expert driver. <IMAGE_TOKENS> Your job is to decide the path of the vehicle for the next five seconds given a <COMMAND>, your <PAST> state, and the image from the vehicle's front camera.

The <PAST> state is the previous four seconds of your trajectory of x-y points in meters spaced one second apart in the following format:
<PAST>(x_-4,y_-4,x_-3,y_-3,x_-2,y_-2,x_-1,y_-1,x_0,y_0)</PAST>

Your output trajectory of the next five seconds should be x-y points in meters spaced one second apart in the same format:

<FUTURE>(x_1,y_1,x_2,y_2,x_3,y_3,x_4,y_4,x_5,y_5)</FUTURE>

Positive x indicates the forward direction, positive y indicates the left direction, and negative y indicates the right direction. Round each number to one decimal place.
You are driving an ATV that has the capability to drive over small plants and on rough trails. Follow the language command as closely as possible.
<COMMAND>Your goal is at (103,92). Proceed there safely.</COMMAND>
<PAST>(-18.3,1.0,-13.7,0.2,-9.2,-0.2,-4.6,-0.2,0.0,0.0)</PAST>
<FUTURE>(4.6,0.4,9.0,1.3,13.3,2.5,17.6,4.1,21.8,6.1)</FUTURE>

Training with Pointing Annotations Our annotations for both of the Gemini auto-labeling runs contain the pixel coordinates of relevant objects and terrain in the scene. In our “with pointing" experiment, we ask the model to predict image points relevant to the language command before it outputs a future trajectory. We track the pixel coordinate chamfer distance between all annotated points and Qwen-predicted points over the course of fine-tuning. The pixel coordinates are normalized between 0 and 1 in the x and y directions. Prior to fine-tuning, Qwen-3-VL-4B achieves a chamfer distance of 0.64. Fine-tuning purely on the language instruction - point pairs results in a chamfer distance 0.40. Training on language instruction, image points, and future trajectory tuples results in a chamfer distance of 0.45. This may indicate that 3D trajectory prediction and pointing are not complementary tasks, but further study of data scale and auto-label quality should be made.

B.3 Closed-Loop Simulation Implementtion

Our simulation assets consist of a 3DGS scene and a gridmap that contains the terrain height. The state of the robot is represented as a current image, current pose, and historical trajectory. During the simulation loop, a render of the 3DGS scene is taken from the pose of the robot and set as the current image. The VLA then uses the current image, historical trajectory, and goal waypoint as input to produce a five second future trajectory in robot-frame BEV space. We pick the point 2 seconds along this trajectory and sample the terrain height gridmap to determine where the robot’s position should be updated. There is no vehicle dynamics model; the robot is in essence teleporting between timesteps. We find that the robot is robust to a simulation timestep between 0.5 and 3 seconds.