Simulation-Based Testing, Validation, and Training with Probabilistic Programming

by

Edward Kim

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Summer 2023

Simulation-Based Testing, Validation, and Training with Probabilistic Programming

Abstract

Simulation-Based Testing, Validation, and Training with Probabilistic Programming

by

Edward Kim

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Sanjit A. Seshia, Co-chair

Professor Alberto Sangiovanni-Vincentelli, Co-chair

Cyber-physical systems (CPS) are increasingly becoming autonomous. Self-driving cars, for example, need to navigate through bustling streets of San Francisco. To avoid the risk of injuries, simulation provides a safe setting to extensively test these systems prior to their deployment. However, as these autonomous systems operate in more complex environments, this poses a challenge as to how to formally model and generate these environments in simulation. In this dissertation, we argue that a domain-specific probabilistic programming language (PPL) can be an adequate formalism to naturally capture the stochasticity and the constraints deriving from physical interactions of these systems with their environments.

The contribution of this thesis is to show how a domain-specific PPL can be effectively used as an environment modeling formalism to test, validate, and train autonomous systems. First, we formalize a machinery to scalably test a system of multi-objectives in parallel simulations with distributions of environments, and summarize the likely causes of those failures in an interpretable manner. Second, we validate whether the identified system failures in simulation transfer to reality, with probabilistic programs as consistent environment models across both simulation and reality. Informed by the validated system failures, we develop algorithms to train components of autonomous systems to be robust against those failures. Furthermore, we devise a personalized algorithm to also train human-CPS (h-CPS), cyber-physical systems that operate in concert with human operators, via simulations in augmented and virtual reality.

Soli Deo Gloria

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I thank my advisors, Sanjit A. Seshia and Alberto Sangiovanni-Vincentelli, for teaching the principles that I treasure and abide by. After all his achievements, Alberto has taught me that the most important thing in life is to just *simply do what one enjoys doing.* This principle has encouraged me to reflect and eagerly search for and pursue research direction that fulfill the values I treasure. By his actions, Sanjit has taught me to *listen* first before I speak. In fact, whenever he and I spoke simultaneously by accident during a discussion, Sanjit would always let me speak first. His principle of listening offered me the space to pitch ideas and be creative.

I thank Prof. Stuart Russell from UC Berkeley and Prof. Daniel Fremont from UC Santa Cruz, for serving in my thesis and qualifying exam committees. I thank Prof. Alvin Cheung for serving in my qualifying exam committee. I would also like to thank Prof. Bjoern Hartmann and Prof. Zachary Pardos for helping me expand the scope of my research to human-computer interaction. I would also like to thank the National Science Foundation (NSF) graduate fellowship for providing me the freedom to shape and pursue my passion.

I thank my collaborators from NASA AMES research center, Corina Pasareanu and Divya Gopinath. My summer internship at NASA in my second year of PhD was fundamental for shaping this thesis. I am also grateful for the privilege of collaborating with industry which exposed me to practical research problems. In particular, it was an eye opening experience to physically experience by skin the process of track testing with American Automobile Association (AAA) and LG Electronics. One of the perks of attending UC Berkeley was to commute to South Bay and discuss research on autonomous systems with developers at the frontier in person. I would like to especially thank Eunsu Ryu, a former principal engineer at Cruise autonomous, for his thoughtful comments as I shape my thesis directions.

Furthermore, I am grateful for colleagues who I had the pleasure to collaborate or interact with. I especially thank my colleagues, Kevin Cheang, Akhil Shetty, and Dayeol Lee, who started and persevered through this PhD program together with me. I would like to also thank Kimin Lee, Abdus Salam Azad, Marcell Vazquez-Chanlatte, Jeongseok Son, Richard Liaw, Shromona Ghosh, Hazem Torfah, Yash Pant, Sebastian Junges, Tommaso Dreossi, Hussein Sibai, Hadi Ravanbakhsh, Markus Rabe, Ankush Desai, Eric Kim, and Ben Caulfield for their kind advice and company. Additionally, I would like to acknowledge my wonderful undergraduates and master students who I advised for over a year: Alton Sturgis, James Hu, Daniel He, Jay Shenoy, Mark Wu, and Kesav Viswanadha.

Lastly, *but most importantly*, I thank my family. I am deeply grateful to my wife, Yuri Cho, who has encouraged, motivated, and inspired me to shape and pursue my vision since undergraduate years. To my parents, it has been my privilege to be raised by a mother and a father with a strong emphasis on education and faith in God. I am deeply grateful for their unconditional sacrifices. I dedicate this thesis to my family.

Chapter 1

Introduction

Over the past decade, machine learning (ML) [6] driven by deep neural networks (DNNs) has contributed to unprecedented advances in cyber-physical systems (CPS). Most prominently, DNNs have enabled these systems to perceive the physical world remarkably well in some cases. This significant advance in perception, combined with control theory, equipped these CPS with increasing autonomy. These autonomous systems operate in complex environments often physically interacting with humans. To name a few, we observe fleets of self-driving cars transporting people in San Francisco [141], navigator bots at airports escorting passengers to their gates [169], bots serving food at restaurants [28], and indoor cleaner bots navigating homes to vacuum and mop floors [122].

In contrast, we witness the rising concern over these growing use of DNNs. Despite their high performance, DNNs have been repeatedly shown to be surprisingly brittle to even imperceptibly small changes in sensor data [81]. In fact, it is very difficult for researchers to understand *why* they are so brittle [4]. The sheer sizes of these DNNs, consisting of large numbers of nodes and weights, have made it extremely challenging to interpret what these models have learned, let alone check their validity. Consequently, we are left in an uncomfortable situation where developers of autonomous systems are uncertain when their DNNs may fail and how their system would behave then. Most concerning is that these DNNs are operating in safety-critical systems like self-driving cars, in which certain important properties like safety must be guaranteed. For instance, in 2022, automakers reported approximately 400 crashes of vehicles with partially automated driver-assist capabilities [93]. These high performing systems, yet with lack of safety guarantees, have raised public safety concerns. This motivates the needs for formal methods to assure the safety of artificial intelligence (AI) and ML-based autonomous systems [156].

Given the urgency of this safety issue, in this thesis, we propose formal techniques to design and analyze autonomous systems or their DNN components in simulation to conform to desired properties. First and foremost, we base our techniques in simulation because there is no risk of physical injury to endanger public safety. And, we can flexibly control environments to examine systems in diverse situations, especially in ones that are rarely encountered in reality to stress test them. Additionally, it is much more cost-effective to reconstruct environments in simulation than via manual physical labor. These cost benefits allow us to scalably and extensively examine DNNs or systems in simulation. On the other hand, in practice, system design and analysis solely based on simulation is problematic due to potential discrepancies between simulation and reality. In particular, synthetically generated sensor data in simulation may result in system failures that are not reproducible with real sensor data captured in reality. Worse, the system may perform without error in simulation but not in reality. Hence, the motivating research questions for this dissertation are:

1. How should we formally search for failures of autonomous systems or their DNN components in simulation and analyze their causes in an interpretable manner?
2. What are efficient ways to validate whether identified failures in simulation transfer to reality?
3. Guided by the results of 1 and 2, how should we effectively (re-)design these systems?

1.1 Thesis Preview

Reflecting the three research questions, this dissertation spans a cycle of system design and analysis as visualized in Fig. 1.1. It formalizes and devises machinery to (i) formally evaluate a system or its components in simulation and intuitively understand likely causes of failures, (ii) efficiently validate whether identified failures in simulation are reproducible in reality, and (iii) update or re-design the failing components of the system via targeted training informed by the results of (i) and (ii).

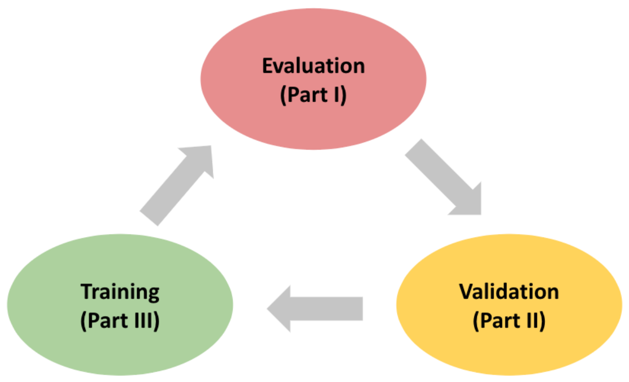


Figure 1.1: Overview of this thesis.

To formulate the problems and provide algorithmic solutions, we integrate theories and ideas from disparate research fields from formal methods [33] to probabilistic programming language (PPL) [68]. In particular, an underlying challenge in our cycle of system design and analysis is to realistically model the increasingly complex environments where autonomous systems like self-driving cars operate. *A unique angle of this thesis is to investigate an existing, adequate formalism to model such environments and address the challenges in utilizing the formalism to evaluate, validate, and train a system.*

1.1.1 Challenges of Modeling Environments of CPS

Autonomous systems like self-driving cars and indoor bots physically interact with humans both outdoors and indoors. These settings incur complexities that are difficult to model

Formalism	Controllable	Dynamic	Reactive	Stochastic	Spatio-Temporal Constraints
Scene Graphs	✓	✓	X	X	✓
GANs, Diffusion LLMs	△	✓	X	✓	✓
Domain-Specific PPLs	✓	✓	✓	✓	✓

Table 1.1: Comparison between existing formalisms to model and generate sensor data in different environments. GANs stand for Generative Adversarial Networks. LLMs stand for Large Language Models. The ✓ indicates satisfaction of a property, whereas X represents non-satisfaction, and △ an incomplete satisfaction.

for several reasons [156]. First, the interactions with humans naturally incur variations, or stochasticity, both in spatial and temporal aspects. For example, when encountering an autonomous vacuum cleaner bot, people may avoid the bot in various ways. Some may walk to the left or right of the bot, another may stop and yield, and others may even step over the bot. These are temporal constraints on how humans may interact with the bot in time, in contrast to any random sequences of actions. Also, prior to the collision avoidance interaction, the initial states (e.g. positions and orientations) of the human and the bot may vary. They may be approaching from a perpendicular or straight head-on directions. These are spatial constraints over the space of random directions and positions. Even for this simple case of collision avoidance with the bot, humans exhibit stochasticity which needs to be modeled. Second, the interactions of these autonomous systems in the physical world naturally abide by physical spatio-temporal constraints. Trivially, the initial positions of the objects cannot be physically intersecting with each other, which is unrealistic. This is a spatial constraint. Also, for the collision avoidance example, the human and the bot need to be on a collision path such that collision is bound to occur if no action is taken on either side. This is a temporal constraint. Hence, to model the complex operating environments of autonomous systems, we need to model stochasticity and constraints over spatio-temporal relations among physical objects.

1.1.2 Existing Environment Modeling Formalisms

Table 1.1 compares the pros and the cons of existing formalisms to model and generate either desired environments in simulation or contents of sensor data. In particular, we compare whether formalisms can precisely model *dynamic* and *reactive* behaviors of objects, and *control*, or generate, environments. Specifically, the *reactive* behaviors refer to being able to interact with a blackbox autonomous system provided for an evaluation, whose behavior is unknown a priori. Also, we compare whether these formalisms can model and generate *stochastic*, or distributions of, environments with *spatio-temporal constraints* over the initial states and behaviors.

Scene graph [31] is a popular graphical formalism to model and generate environments. Its nodes represent objects (e.g. pedestrian, car) and edges represent spatial relations between the connected objects (e.g. a pedestrian is *in front of* a car). In conjunction with a simulator, scene graphs can generate static sensor data in desired environments. Various extensions of scene graphs have been introduced to also model and generate time series data [31]. However, scene graphs or their extensions cannot model either distributions of environments or reactive behaviors of objects in response to a given blackbox system.

Natural languages like English have been increasingly popularly used to generate sensor data through the advances in text-to-image and text-to-video algorithms backed by generative adversarial networks (GANs) [110], diffusion [109], and large language models (LLMs) [5]. Instead of relying on a simulator, these generative models can directly render time series sensor data with environments specified using natural languages, with which we can model and generate dynamic and stochastic behaviors with spatio-temporal constraints. However, it is often difficult to generate sensor data with desired environments using these generative models, requiring extensive prompt engineering. Furthermore, the generated sensor data are not reactive because these algorithms do not take into account the actions of a blackbox autonomous system under evaluation when generating sensor data. Although the dynamics of these autonomous systems and their policies may be modelled using neural networks, the cost of training such networks and the uncertainties in their outputs stand as practical challenges.

The existing general probabilistic programming languages such as PROB [68], Church [66], and BLOG [119] do not provide domain-specific supports necessary to model and generate complex physical environments. These languages primarily focus on *inference* rather than *generation* for testing purpose. In particular, it is challenging to model spatio-temporal constraints over distributions with these languages, let alone correctly sampling from the distributions to satisfy such constraints. In contrast, domain-specific PPLs [60, 115, 54] do support all five functionalities in Table 1.1 that are necessary to model and generate complex physical environments in a simulator.

1.1.3 Environment Modeling with Domain-Specific PPLs

Note that we emphasize the "domain-specific" aspects of PPLs. To model and generate *physical* environments, the environment formalism should support domain-specific functionalities to (i) *model* pertinent spatio-temporal constraints and (ii) *correctly sample* an environment from a distribution with such constraints. If we merely specify distributions with no spatio-temporal constraint, then we may generate unrealistic environments where, for example, two vehicles are instantiated physically intersecting with each other. Hence, PPLs with these domain-specific supports are desirable for modeling the operating environments of CPS.

Various domain-specific probabilistic programming languages have been employed to define generative models for objects and scenes, such as Quicksand [100] and Picture [99]. However, they do not provide dedicated syntax or semantics to model geometry or dynamic behaviors. Note, for instance, Picture was solely used for inverse rendering and not data generation. In this thesis, we use a domain-specific PPL called Scenic [60, 59], whose syntax and semantics are designed to model geometric and dynamic behaviors. We will explain its details later in the Background (Sec. 2). The primary advantage of Scenic over these languages lies in its domain-specific syntax and semantics. This permits concise representation of complex environments and enables specialized sampling techniques with emphasis on the *generation* of scenarios.

1.1.4 Challenges of Testing, Validation, and Training of CPS with Probabilistic Programming

It takes a strong discipline to rigorously test, validate, and train or design a system to satisfy certain properties, i.e. requirements. In this thesis, we build upon the discipline of formal methods [33], but the challenges arise from incorporating the PPL formalism to techniques in formal methods. Before we expand on the challenges, we justify our choice of discipline. Formal methods is a promising approach to rigorously design, analyze, and verify certain properties like safety based on mathematical reasoning. This rigor starts with specification, i.e. the process of encoding unambiguous properties, which the system must abide by, into a machine-readable formalism. Given the specification and the models of both the system and its operating environment, formal methods provides techniques to *prove* that the system satisfies the specification or, if not, provide a *counterexample*. The identified counterexamples can be used to better design the system. Note that to design, analyze, and verify properties of a system, formal methods require a model of its environment.

The challenges we address in this thesis arise from incorporating a PPL formalism to model environments while devising machinery that are compatible with formal methods. Although this effort has previously been initiated [58], there are still many open questions to be resolved which we expand on in the following.

Testing

We focus on the challenges pertaining to the scalability and the interpretability of system testing. In regards to scalability, Although there have been prior work which develop mechanisms to incorporate Scenic domain-specific PPL to formal methods [60, 46, 115], their support for scalable testing has been limited. First, they do not provide a formalism to specify multiple system properties with differing priorities, with a partial order. For example, a self-driving car should not only transport a passenger to a destination but also abide by traffic rules, and, most importantly, avoid collision. Second, given a distribution of scenarios modelled with Scenic, how to intelligently search, or sample, the distribution to identify any counterexamples with respect to scaled system properties with varying priorities is not addressed. Finally, the machinery to scale, or speed up, the testing with parallel simulation to support the intelligent search of counterexamples for the scaled number of system properties is also missing.

In addition to identifying system failures, it is crucial to analyze and understand *why* the failures occur in order to accurately debug the system. For example, it would be helpful to understand that a self-driving car tends to crash when taking an unprotected left turn under a rainy weather due to a perception failure. How can we devise an algorithm which automates the analysis and succinctly outputs its summary in an interpretable manner? In particular, we focus on using the Scenic formalism as a means to symbolically summarize the causes of failures.

Validation

Once we identify system failures via testing in simulation, another relevant problem is to *validate* whether the failures transfer to, or are reproducible, in reality. For brevity, we use *sim-to-real* validation to denote this problem of validating system performance from simulation to reality. The necessity for this sim-to-real validation derives from a number of discrepancies between simulation and reality. To list a few, for instance, in autonomous driving domain, the dynamics models of vehicles in simulation may not represent all the physical variables (e.g. tire friction, air drag) that affect the dynamics in reality. The modeled behaviors of vehicles and pedestrians in simulation may not be representative of the behaviors observed in reality. Also, the synthetics sensor data (e.g. RGB images, LiDAR point cloud) from simulation may differ from the real data collected with sensors in reality, which may induce system behaviors and, therefore, invalidating the results of testing in simulation.

An approach for sim-to-real validation is to physically reconstruct failure environments in the real world to validate the system performance. In autonomous driving domain, for example, track testing [61] is a well-established means for sim-to-real validation. The strength of this approach is that it takes into account many of the aforementioned discrepancies between simulation and reality, simultaneously. Yet, its scalability, in terms of the number of environments that can be physically reconstructed, is severely limited due to the labor-intensive nature. Hence, it is crucial to carefully select which test environments to reconstruct. Can we effectively leverage simulation, with Scenic as an environment formalism, to generate test environments for track testing such that they reproduce system failures in reality? This is one of the two challenges relating to sim-to-real validation we focus on in this thesis.

Although track testing jointly considers different sources of discrepancies between simulation and reality, its scalability is severely restricted. To scale the validation process, we restrict our attention to a single dimension of the discrepancies and focus on relaxing the need for physical reconstruction of environments. In particular, we focus on sim-to-real validation for sensor data discrepancy with application to validating performances of DNN-based perception. The challenge here is to algorithmically compare the *contents* of environments represented in synthetic versus real sensor data. For example, suppose we identify in simulation that an autopilot fails to perceive another vehicle abruptly cutting into its lane. If we can identify a set of real sensor data with the same contents of such cut-in behaviors, then we can test the perception model on those real data for sim-to-real validation, without having to physically reconstruct such environments. Given that massive amounts of real sensor data are collected and labelled [41, 32], the challenge is to use Scenic as an environment formalism to query the labelled dataset to output a subset of real sensor data with contents of interest.

Training

After we identify and validate environments which result in system failures, we need to debug the components of the ML/AI-based CPS. These failure-inducing environments can be encoded as probabilistic programs. Previous literature have utilized these probabilistic programs as generative models of sensor data for training data augmentation, with application to DNN-based perception [45, 57]. Yet, there is an insufficient debugging technique for other potentially ML-based components of autonomous systems, given Scenic as an environment formalism. Hence, we focus on debugging deep reinforcement learning (RL) [165] algorithms which are increasingly utilized for planning and control. Specifically, we investigate how to use Scenic to effectively and efficiently fine-tune these RL algorithms in either online or offline.

Furthermore, leveraging the techniques we develop in this thesis, we also focus on training human cyber physical systems (h-CPS), which refer to autonomous systems with human in the control loop. For example, a fork truck maneuvered by a human driver, a drone whose trajectories are designated by a human, or a soldier on a battle field wearing a helmet with a display which visualizes real-time information of surroundings are h-CPS. In addition to AI/ML-based components, the performances of these h-CPS depend on the abilities of humans who may have varying degrees of control. We focus on training the humans to better control the systems to better achieve tasks. To safely train humans, we use immersive augmented and virtual reality (AR/VR) [123]. The challenge is to personalize the training environments, modelled with Scenic, to each human's learning speeds to maximize one's learning.

1.2 Contributions

This thesis contributes a collection of algorithms, which incorporates a domain-specific PPL as an environment modeling formalism, in order to (i) scalably test and analyze the causes of system failures in an interpretable manner, (ii) efficiently validate the simulation test results in reality, and (iii) systematically train these systems informed by the test and validation results.

1.2.1 Scalable and Interpretable Testing

We propose two complementary algorithms. First, in the context of adversarial ML [81] and formal methods [33], we develop a toolkit which formally specifies multiple system properties with varying priorities [183]. It efficiently searches, or samples, from distributions of environments modeled with Scenic to identify environments that violate the system properties with parallel simulations. Second, in the context of explainable AI [4], we devise a system-level debugging algorithm which analyzes system failures and makes a novel use of Scenic to succinctly summarizes the causes [89].

1.2.2 Sim-to-Real Validation of System Performance

We contribute two algorithms for sim-to-real validation. The first algorithm can validate a full scale system with limited scalability, while the second algorithm validates a DNN-based perception models in a scalable way. The first algorithm generates effective test environments for track testing [61]. This algorithm extensively tests an autonomous system in simulation with distributions of environments modeled using Scenic and selects environments which would likely induce system failures in reality. Given a big, labelled real sensor dataset, the second algorithm uses Scenic as a query language to model environments and retrieve sensor data which matches the environment description [91].

1.2.3 Failure-Informed Targeted Training

Using the generative aspects of Scenic, we develop online and offline training techniques for RL algorithms which are increasingly adopted for CPS [13]. Furthermore, we contribute a personalized algorithm in augmented and virtual reality (AR/VR) [123] to train h-CPS [90]. In particular, this h-CPS training algorithm focuses on training skills to humans so they can better control the systems and more efficiently achieve tasks. Given a set of skills to train and a corresponding set of probabilistic programs modeling training environments, the algorithm models the human's knowledge state and personalizes, or individually adapts, training environments in AR/VR accordingly to maximize learning. This algorithm contributes to computational interaction [187], an intersection of formal methods [33] and human-computer interaction (HCI) [77].

1.3 Thesis Outline

This thesis is divided into three major parts mirroring the contributions.

1.3.1 Preliminaries

We explain the background necessary to fully appreciate the thesis in Ch. 2. Specifically, we introduce Scenic which is utilized throughout this dissertation, as well as VerifAI [46] which is a toolkit for design and analysis of AI/ML-based CPS and supports Scenic as an environment formalism.

1.3.2 Part I: Scalable and Interpretable Testing

Chapter 3 describes our toolkit which enables scalable testing of an autonomous system. Chapter 4 explains our algorithm which analyzes system failures and summarizes their causes as probabilistic programs.

1.3.3 Part II: Sim-to-Real Validation of System Performance

Chapter 5 focuses on effective test case generation for track testing to validate autonomous systems in reality. Chapter 6 describes our algorithm, which uses Scenic as a query language, to retrieve real sensor data provided a big, labelled dataset to validate DNN-based perception models.

1.3.4 Part III: Failure-Informed Targeted Training

Chapter 7 expands on our techniques to train, or fine tune, deep RL algorithms in either online or offline. Chapter 8 describes our h-CPS training algorithm in AR/VR.

1.4 Bibliographic Notes

The purpose of bibliographic notes is to provide historical contexts, or related literature, for each chapter of this thesis. For this bibliographic note, however, is slightly different in that we acknowledge our collaborators and funding organizations that contributed to the works. We acknowledge that texts, figures, and tables from joint papers with our collaborators are used or adapted for this thesis. We would like to thank the National Science Foundation (NSF) Graduate Fellowship program. It provided us the freedom to pursue our interest and passion.

1.4.1 Part I

The idea to devise a machinery for scalable testing (Ch. 3) is inspired by the VeHICal NSF Cyber-Physical Systems (CPS) Frontier project [155] – in particular, in our interactions with Kesav Viswanadha, Francis Indaheng, Tommaso Dreossi, Daniel Fremont, Shromona Ghosh, and Xiangyu Yue. We would also like to thank Prof. Pravin Varaiya, Alex Kurzhanskiy, Akhil Shetty, and Mengqiao Yu from UC Berkeley for sharing their insights, which helped us shape our research. The subsequent work (Ch. 4) on an interpretable analysis of system failures is an outcome of a summer internship at NASA AMES Research center in collaboration with Corina Pasareanu and Divya Gopinath.

1.4.2 Part II

Our work on test case generation for track testing (Ch. 5) is the result of industrial collaboration with American Automobile Association[1] (AAA) and LG Electronics. We acknowledge our contributors from Daniel Fremont and Yash Pant from UC Berkeley, Atul Acharya, Xantha Bruso, and Paul Wells from AAA, and Steve Lemke, Qiang Lu, and Shalin Mehta from LG Electronics. Our experience of track testing has informed us the labor intensive aspect of track testing which served as a motivation for Ch. 6. We thank Eunsu Ryu, a former principal engineer at Cruise, who advised us to leverage labelled, massive data that AV companies have been collecting. This influenced us to formulate sim-to-real validation problem as a data query problem. We acknowledge the contributions from our collaborators, Jay Shenoy, Sebastian Junges, and Daniel Fremont from UC Berkeley for Ch. 6.

1.4.3 Part III

The case study on the application of a PPL to train reinforcement learning agent (Ch. 7) is in collaboration with Abdus Salam Azad and Kimin Lee from UC Berkeley. The subsequent work on training humans for psychomotor skills with PPL (Ch. 8) spawned from interactions with Prof. Zachary Pardos in the education department and Prof. Bjoern Hartmann in the electrical engineering and computer sciences (EECS) department at UC Berkeley. We also thank the contributions from undergraduates from UC Berkeley's extended reality club, Alton Sturgis and James Hu, who helped implementing the training system in virtual reality.

[1]Specifically, we collaborated with AAA's Northern California, Nevada and Utah (NCNU) division

Chapter 2

Background

2.1 Scenic: Probabilistic Scenario Modeling Language

Scenic [60, 59] is a domain-specific probabilistic programming language to model environments and to generate the environments of CPS in simulation. The language provides the following key features that make it applicable across domains as shown in Fig 2.1: (a) *modeling* distributions of environments with physical constraints, and (b) *sampling* from the distributions with the physical constraints.

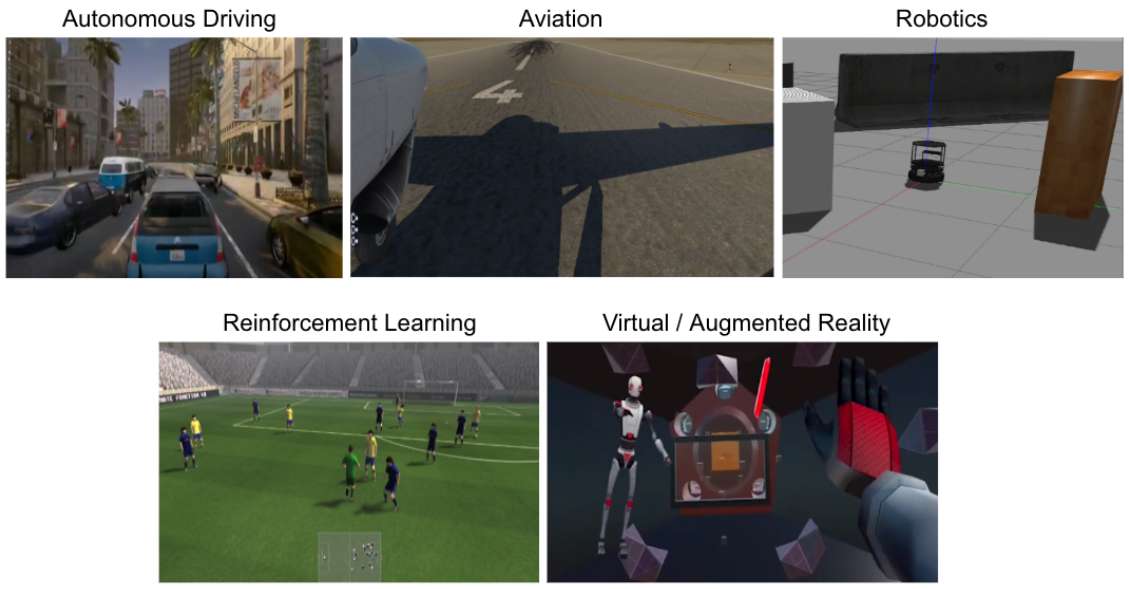


Figure 2.1: Examples of various application domains and simulators Scenic is interfaced to

2.1.1 Definition of Scenarios

So, what is an environment? In this thesis, we denote an environment as an *abstract scenario*, which is a distribution of *concrete scenarios*. An abstract scenario consists of (i) objects, (ii) a distribution of initial states, and (iii) a distribution of behaviors. The *objects* may represent either physical objects, e.g. autonomous systems or humans. The *state* is defined using semantic features of these objects, e.g. positions, orientations, and colors of physical objects and weather conditions and time. An abstract scenario consists of a distribution of initial states, e.g. time may vary uniformly randomly from 8am to 12pm. A distribution of a *behavior* is defined as a mapping from a history of states to a distribution of actions, where an *action* is an instantaneous operation executed by an object like setting a throttle or a steering angle. A behavior, therefore, can maintain a memory. A concrete scenario is an instantiation of an abstract scenario, with concrete values assigned to the initial state and the behaviors mapping a history of actions to a concrete action. In the rest of the thesis, we will simply denote concrete scenarios as scenarios.

2.1.2 Intuitive, Probabilistic Modeling with Physical Constraints

Scenic is a programming language embedded in Python, one of the most popular programming languages. Its intuitive and probabilistic syntax blends Python syntax with simple keywords in English to help users, even non-programmers, to easily model, interpret, modify, and communicate complex scenarios with others. First, Scenic's concise notation based on English makes it easy to learn, write, and interpret Scenic programs. Second, the probabilistic language helps users efficiently model the stochasticity and uncertainty in the real world. For example, there are various ways a lane change behavior on a road can occur. Rather than writing a scenario program for each variation, Scenic helps users to model and generate a *distribution* of scenarios with a single Scenic program with *realistic physical constraints*. To concretely demonstrate the intuitive and probabilistic aspects of the language for example in the autonomous driving domain, we refer to an example Scenic program in Fig. 2.2 modeling a distribution of scenarios, where a badly parked car pulls into a road as another car, which we denote by `ego`, approaches from behind as visualized in Fig. 2.3.

First, note the intuitive syntax highlighted in yellow that resembles the natural English. This syntax helps users model the geometric spatial relations among objects in the initial state. In line 9, `ego` is uniformly randomly placed on a `lane`. This `lane` refers to any lanes in the map of the simulated world. In line 12-15, we model a distribution of initial states of the badly parked car. In line 14, the `parkedCar` is positioned left of a `spot` by 0.5 meter. This `spot`, in line 12, is uniformly randomly sampled from a `visible curb`, i.e. intersection of a curb and the `ego`'s view cone. In line 15, the orientation of the `parkedCar` is perturbed by `badAngle` with respect to the traffic flow direction at the `parkedCar`'s position. This `badAngle` is uniformly randomly sampled from a range of 10 to 20 degrees with either positive or negative sign (i.e. right or left orientation). Furthermore, we can model declarative constraints over spatio-temporal constraints in Scenic. In line 18, we specify a spatial

```
 1 behavior EgoBehavior(safety_distance, target_speed=25):
 2     try:
 3         do FollowLaneBehavior(target_speed=target_speed)
 4     interrupt when withinDistanceToObjInLane(self, safety_distance):
 5         take SetThrottleAction(0), SetBrakeAction(1)
 6     interrupt when collisionOccur():
 7         terminate
 8
 9 ego = Car on lane,
10         with behavior EgoBehavior(safety_distance=10)
11
12 spot = OrientedPoint on visible curb
13 badAngle = Uniform(1, -1) * Range(10,20) deg
14 otherCar = Car left of spot by 0.5, # meter
15             facing badAngle relative to roadDirection,
16             with behavior PullIntoRoad()
17
18 require 10 < (distance from ego to parkedCar) < 20 # meters
19 require always (ego can see otherCar)
```

Figure 2.2: A snippet of an example Scenic program modeling a distribution of badly parked car scenarios.

constraint over the initial state that distance from `ego` to `parkedCar` is less than 20 meters. In line 19, we model a temporal constraint that ego must be able to see the `otherCar` during the scenario at all time.

Scenic further provides syntax and semantics for users to specify the interactive behaviors of objects. `egoBehavior` assigned to `ego` is defined in line 1-7, using Scenic's try/interrupt block. The `egoBehavior` by default uses `FollowLaneBehavior` in the `try` block. Note that the `do` syntax is used to invoke a behavior. However, if the interrupt condition in line 4 is satisfied, then Scenic pauses the default behavior in the try block and simultaneously executes both the actions which disengages the throttle and applies full brake. The interrupt condition is defined using a pre-defined helper function, `withinDistanceToObjInLane()`, which checks whether there is another object within the given safety distance on the `self`'s lane. Here, `self` refers to the object that is executing the behavior, which is `ego` in this case. Once the interrupt condition no longer satisfies, then the paused default behavior in the try block resumes.

Note that there can be multiple interrupt conditions to assign different *priorities* over interaction conditions. If there are multiple interrupt conditions, then the conditions stated lower have higher priorities. Hence, the second interrupt condition in line 6 has a higher priority than the one in line 4. This means that if both interrupt conditions are satisfied, the behavior or action(s) of the higher priority condition will be executed. Finally, Scenic

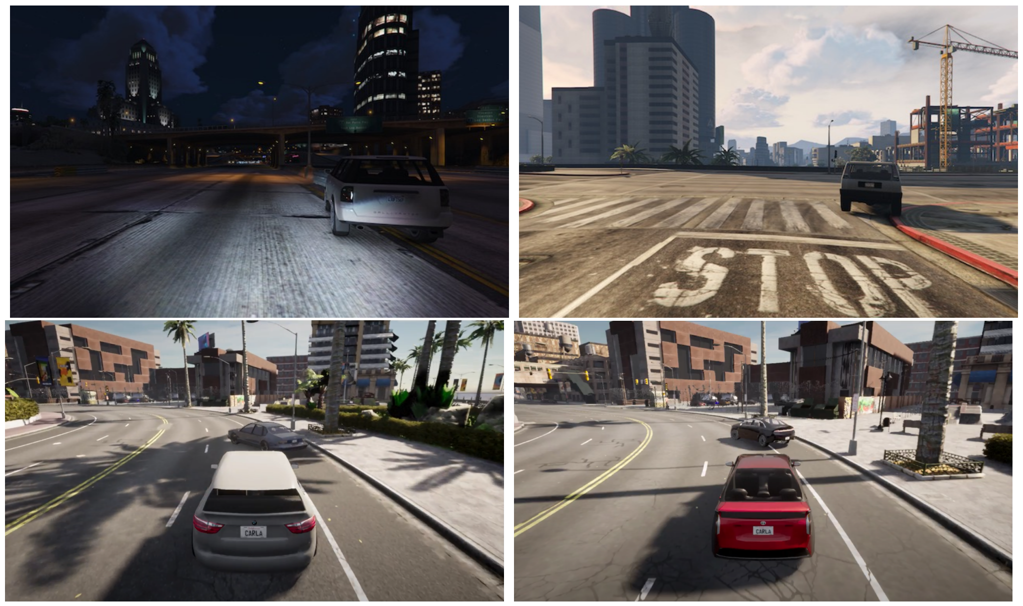


Figure 2.3: Scenes generated from the Scenic program in Fig. 2.2 in GTA-V [63] (top images) and in CARLA [44] (bottom images) simulators

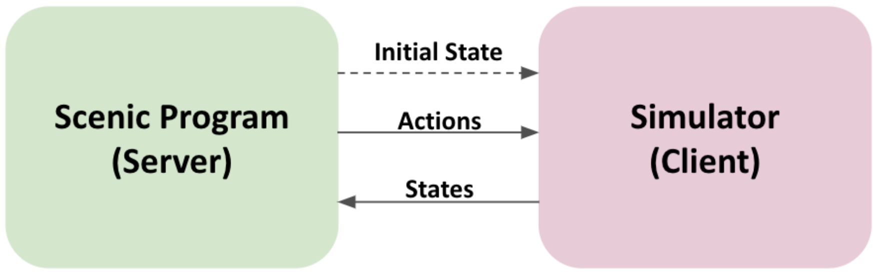


Figure 2.4: Overview of interactions between Scenic and a given simulator

provides a syntax, `terminate` to end the simulation. In line 6-7, the simulation is to terminate when a collision occurs.

2.1.3 Sampling from Distributions with Constraints

In order to *generate* scenarios in simulation, Scenic provides domain-specific support to correctly *sample* scenarios from a distribution with physical constraints, such that the samples satisfy the constraints. To efficiently sample scenarios, Scenic uses internal pruning heuristics based on an analysis of the constraints (refer to [60]). The architecture of how Scenic interacts with a given simulator to generate reactive scenarios is shown in Fig. 2.4. First, the Scenic program and the simulator forms a server-client relation to initiate communication. Scenic program samples a scene, i.e. initial state, which contains information such as initial position, orientation, and color of objects, as well as weather condition, time of day, etc. The sampled scene is sent to the simulator which instantiates the scene, as visualized in Fig. 2.4 with a dotted line. The simulator updates the state of the world (at this point, the world state is the same as the initial state) and sends it to the Scenic program. Given the world state, the Scenic program samples action(s) per object to simulate for 1 simulation timestep and send them to the simulator. Users can flexibly determine the duration of this timestep. The simulator simulates the action(s) per object for 1 simulation timestep, updates the state of the world, and sends the world state back to the Scenic program. This cycle of communication continues either until the scenario completes.

2.2 VerifAI Toolkit

Scenic provides a modeling language with the ability to generate concrete scenarios. However, the design and verification of AI based systems requires many more algorithmic components. These are implemented in VerifAI toolkit [46]. The architecture of VerifAI is shown in Fig. 2.5. VerifAI takes as inputs the (i) system model, (ii) environment model, and (iii) property. A property is a requirement that the system should satisfy. For the system model, an implementation of the system is used, e.g. autopilot software. The environment model can be specified as a Scenic program. And, the property of the system can be flexibly encoded using an objective function or temporal logic [103]. In the following we explain each component (highlighted with colors) in VerifAI.

2.2.1 Semantic Feature Space

A semantic feature space consists of a set of features and their ranges. In VerifAI, one can either directly define the semantic feature space or use Scenic to do so. If a Scenic program is given, VerifAI analyzes the structure of the program and extracts a semantic feature space (colored in green in Fig. 2.5). For example, the Scenic program may specify distributions over features such as positions, orientations, and colors of objects as well as weather conditions and time of day. These feature distributions can often be *dependent* on others.

2.2.2 Search (Sampling)

The goal of this search component (colored in blue) is to intelligently search for system failures by efficiently sampling from the semantic feature space. VerifAI supports both *active* and *passive* samplers. The active samplers sample a scenario based on the history of sampled scenarios and the corresponding system evaluation results. In contrast, the passive samplers sample a scenario regardless of this history. For passive samplers, VerifAI implements uniform random and Halton [74] samplers. For active samplers, VerifAI implements

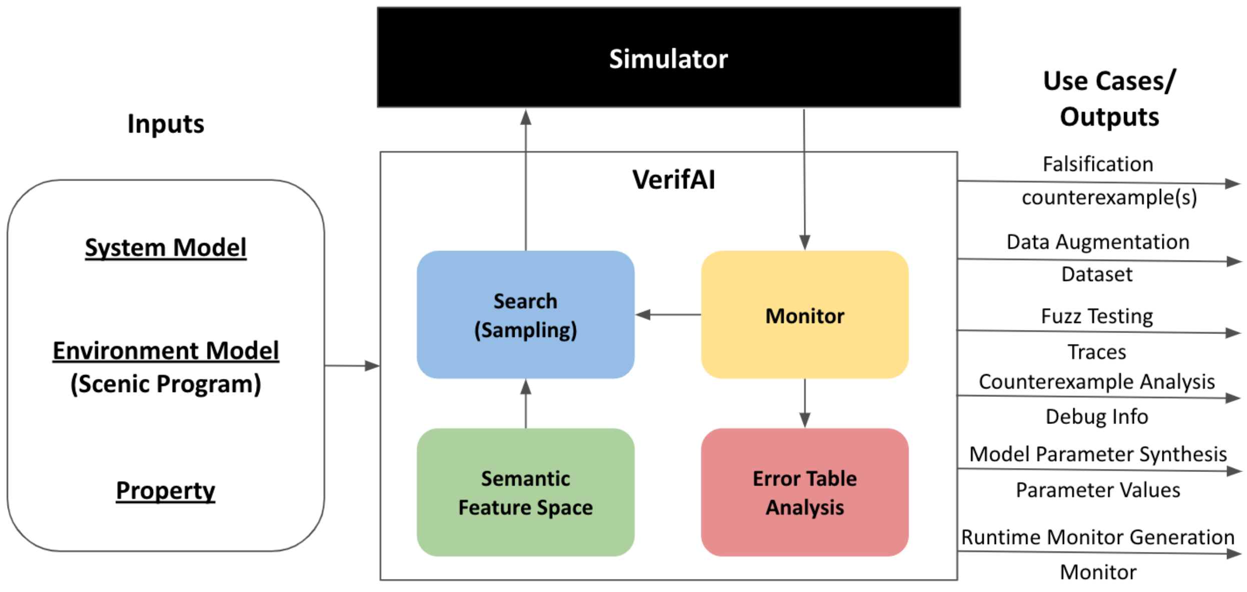


Figure 2.5: VerifAI Architecture

cross-entropy [148], simulated annealing [92], bayesian optimization [55] samplers. When using Scenic with VerifAI, users can designate feature variables in the Scenic program as external parameters to be sampled by VerifAI. Also, in VerifAI, users can select a sampler of their choice or import their own samplers. Once the sampler samples a scenario, it is sent to the simulator which tests the system in the sampled scenario as shown in Fig. 2.5.

2.2.3 Monitor

During scenario generation in the simulator, the monitor (colored in yellow) records the system trace and the sampled scenario. After the scenario completes, the monitor evaluates the system using the input property. The monitor shares the information on (a) the sampled scenario and (b) the system evaluation result in the scenario to the sampler (blue) and the error table (red).

2.2.4 Error Table Analysis

The error table (colored in red) records the list of sampled scenarios and corresponding system evaluation results. The error table provides algorithms to support analysis, such as principal component analysis (PCA) [20] and k-means clustering [160] algorithms. These algorithms can be used to identify any patterns among the semantic features, which likely induce system failures, to better understand and debug the system.

2.2.5 Use Cases

VerifAI serves multiple use cases which are shown on the right side of Fig. 2.5. First use case is *falsification* which is to search for scenario(s) which violate a given property [183]. Second use case is data augmentation [45, 57]. We can collect sensor data from the failure scenarios with the correct labels from the simulator. This way, we can systematically augment training data to help train a system to be more robust to the identified failures. Leveraging the generative aspect of Scenic, VerifAI can also generate variations of scenarios to fuzz test a system to output system traces [46]. As mentioned in above in the error table analysis section, VerifAI can analyze any similarities or patterns in the scenarios which induce system failures, or counterexamples in short, to more systematically debug a system [89]. And, VerifAI can also be used to generate system monitors which can detect system failures in runtime [152].

Part I

Scalable and Interpretable Testing

Chapter 3

Parallel and Multi-Objective Falsification

There are multiple interrelated challenges when conducting system-level testing of autonomous systems. First and foremost, we need to formally specify potentially multiple objectives of these autonomous systems with different priorities. For example, a self-driving car should not only transport people from one location to another, but also abide by traffic rules and, most importantly, avoid collision. This formalism should effectively guide the falsification with respect to stochastic operating environments of autonomous systems. Furthermore, an efficient search (sampling) algorithm is necessary to explore distributions of environments and find diverse scenarios where the systems violate various priorities of objectives. Finally, it is unclear how to integrate the solutions for the aforementioned interrelated challenges with parallelized simulations to expedite falsification.

In this chapter, we jointly address these challenges to devise a machinery for *scalable* system-level testing of autonomous systems, with *parallel* falsification of *multi-objectives*. Specifically, we extend the VerifAI toolkit to (i) formally specify multi-objective specification with priority ordering, (ii) efficiently search a distribution of scenarios modelled as a Scenic program using a multi-armed bandit (MAB) algorithm, and (iii) parallelize falsification, running multiple simulations in parallel, to expedite the process [183].

3.1 Methodology

3.1.1 Parallel Falsification

A bottleneck of simulation-based testing is the generation of the simulation trajectory in the simulator. A simulation trajectory is a sequence of timestamped states which may include information such as positions and orientations of objects at every timestep. Depending on the complexity of the scenario description and the computation required by the simulator, it may take more than a minute to simulate each scenario. This bottleneck affects the scalability of testing in simulation, limiting the number of scenarios to test per day.

We present an improvement on this pipeline by parallelizing it using the Python library

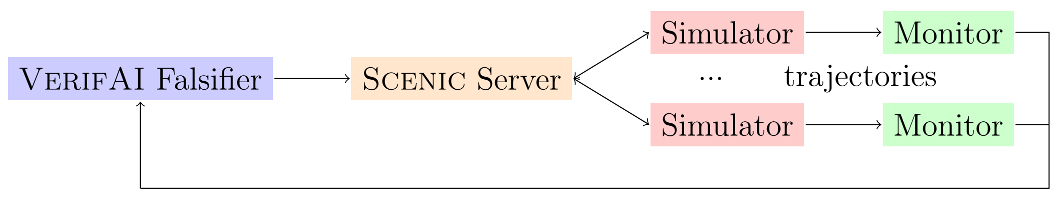


Figure 3.1: Parallelized pipeline for falsification using VerifAI.

Ray [120], which encapsulates process-level parallelism optimized for distributed execution of compute-intensive tasks. Fig. 3.1 illustrates the new setup. We instantiate multiple instances of the simulator and open multiple Scenic server connections from VerifAI to the those instances. The Scenic server sends a sampled scenario to each simulator instance for simulation. The simulation trajectory collected from each simulation instance is evaluated using the monitor in VerifAI. The simulation result is then sent back to VerifAI's falsifier, or the sampler, to update its internal sample space. We aggregate the results of these simulations into a single error table documenting all the safe and failure scenarios found during falsification.

Our improved pipeline also includes modifications to the existing sampling mechanism in VerifAI. Previously, VerifAI invokes its `nextSample` API to sample the next scenario, which is incompatible with parallel falsification. The reason is that this API, which consists of both scenario sampling and internal sample space update, forces all simulations in parallel to complete first in order to update its internal sample space. Note that this internal update is necessary for active samplers which take into account the *history* of sampled scenarios and their simulation results. In our improvement, we replace the `nextSample` API with `getSample` and `update` APIs. This decomposition relaxes the previous bottleneck to enable parallel falsification.

3.1.2 Multi-Objective Falsification

There are typically many different metrics of interest for evaluating autonomous systems. For example, there are many well-known metrics used in the autonomous driving community to measure safety: no collisions, obeying traffic laws, and maintaining a minimum safe distance from other objects, among others [191]. It is also natural to assert priority, for example, that it is more important to avoid collisions than to follow traffic laws. The existing VerifAI toolkit only allows specifying a single objective or property which makes it difficult to express multi-objectives with a priority order. We now discuss how to specify these metrics and their relative *priorities*.

Specification of Multiple Objectives Using Rulebooks

We adopt an existing *rulebook* formalism Censi et al. [29] to model multiple objectives, or properties, with different priorities. First, a property defines a requirement that a system should satisfy. Formally, let ρ be a function mapping a simulation trajectory, x, generated

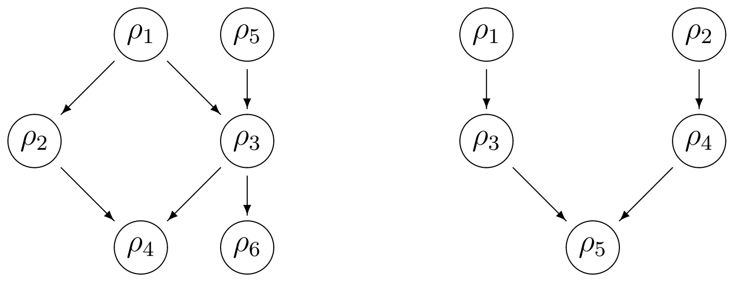


Figure 3.2: Left: example rulebook over functions $\rho_1 \ldots \rho_6$ [29]. Right: graph G used in experiments.

by SCENIC or VERIFAI to a vector of concrete values, where $\rho(x)$ is the application of ρ to x and $\rho_j(x)$ is defined as the *value* of the j-th property. For this value, note that a real number can be used, for instance, with robust semantics of signal temporal logic (STL) [116]. Censi et al. [29] have developed a way to specify preferences over these metrics using a *rulebook* denoted by $\mathcal{R}$ – a directed acyclic graph (DAG) where the nodes are the metrics and a directed edge from node i to node j means $\rho_i(x)$ is more important than $\rho_j(x)$. We denote this using the $>_R$ operator, e.g. $\rho_i >_R \rho_j$, which is used in [29].

The left DAG in Fig. 3.2 shows an example of a rulebook over six metrics $\rho_1, \ldots, \rho_6$. In this example, we can make several inferences, such as ρ_1 is more important than ρ_3, ρ_3 is more important than ρ_4, and ρ_5 is more important than ρ_3. However, there are also many pairs of objective components that cannot be compared; for example ρ_1 and ρ_5. Nevertheless, we would like to have a way to order objective vectors to know which values are maximally violating the specification during active sampling. Therefore, the rulebook $\mathcal{R}$ allows a *preorder*, $\succ$, over the objective vectors. Censi et al. defines the preorder as follows.

$$\rho(x_1) \succ \rho(x_2) \triangleq \forall i. \left((\rho_i(x_1) < \rho_i(x_2)) \implies \exists j.((\rho_j >_R \rho_i) \wedge (\rho_j(x_1) > \rho_j(x_2))) \right)$$

Multi-Objective Active Sampling

Given a rulebook $\mathcal{R}$ and a preorder $\succ$, the goal of falsification is to search for the worst counterexample scenario by sampling scenarios, ideally violating all the properties. When performing active sampling (refer to Ch. 2.2) to search for unsafe test inputs, we need a specialized sampler to support having multiple objectives to guide the search process. The samplers previously available in VERIFAI focus either on exploration of the search space or entirely on exploitation of a subset of the search space to find unsafe inputs. We present a sampler that balances these and builds up increasingly-violating counterexamples in the multi-objective case.

The Multi-Armed Bandit Sampler. We implement a *multi-armed bandit* (MAB) sampler by adapting the algorithm from Carpentier et al. [24] to our falsification setting; the idea of this sampler is to balance the trade-off between exploitation and exploration. To understand the motivation for the sampler, we first look at the formulation of the MAB problem [98]. Consider a bandit which has multiple lotteries, or “arms", to choose from, each being a random variable offering a probabilistic reward. The bandit does not know ahead of time which arm gives the highest expected reward. In this setting, the objective is to efficiently sample the arms while maximizing the average earned reward during the sampling process. In other words, one needs to explore different arms to identify their associated expected rewards while exploiting this knowledge to earn higher average reward.

Carpentier et al. [24] present the Upper Confidence Bound (UCB) algorithm that achieves this goal. To briefly explain, the algorithm samples the arm, j, that maximizes a quantity, Q_j, which is defined as:

$$Q_j = \hat{\mu}_j + \sqrt{\frac{2}{T_j(t-1)} \ln\left(\frac{1}{\delta}\right)} \tag{3.1}$$

where t is the trial number, $T_j(t-1)$ is the number of times the jth arm is sampled until $(t-1)$th trial, δ is a confidence parameter, and $\hat{\mu}_j$ is the observed reward of arm, j.

Qualitatively, this formulation balances between exploitation of the reward distribution learned so far (the first term) and exploration of seldom-sampled arms (the second term). We adapt this algorithm to our falsification setting. Suppose a rulebook $\mathcal{R}$, its preorder $>$, a system, and a simulator are provided. We discretize the scenario space into N number of buckets which we equate to the notion of arms in the MAB problem. By selecting a bucket, we (a) uniformly randomly sample a scenario from the bucket, (b) test the system under the scenario in the simulator to collect a simulation trajectory x, and (c) compute $\rho(x)$. We equate this $\rho(x)$ to the reward in the MAB problem. The goal of falsification is to search for scenarios which results in the worst $\rho(x)$ value according to the preorder relation, ideally violating all the properties in the rulebook.

In the following, we describe our adaptation of the Upper Confidence Bound (UCB) algorithm developed by Carpentier et al. [24] to our falsification setting. Before we do, we first define relevant terminologies for a succinct explanation. Recall from the Background (Ch. 2.1) that a scenario space is equivalent to a semantic feature space. We denote a semantic feature space consisting of k number of features to be $D = D_1 \times D_2 \times ... \times D_k$, where D_i is the domain of the ith feature. We formalize a sampled scenario as $f = [f_1, f_2, ..., f_n] \in D$, where f_i represents a concrete value for the i feature. With these denotations, we explain our adaptation below.

Setup

1. Evenly split the domain of each feature, D_i, into N number of domains for all k features.
2. Initialize matrix T of size $k \times N$ to keep track of the number of trials that each bucket is selected, where its column index corresponds to features and row index to the features' N buckets. T_{ij} element records the number of trials that the ith feature selects its jth bucket.
3. Initialize a dictionary, d, to keep track of which buckets result in high rewards. First, we abstract a vector-valued ρ reward to a boolean vector reward, b. We convert non-

boolean values in the reward vector to booleans by thresholding the values[1]. The dictionary, d, maps the abstracted worst boolean reward, b, to a matrix M_b of size $k \times N$. $M_{b,ij}$ records the number of trials where the ith feature selecting its jth bucket yields the reward, b. Note that there can be multiple worst counterexamples due to the preorder relation. As we will see in the **Sampling** stage below, the boolean abstraction of the reward helps us better bookkeep which buckets yield desired rewards. Otherwise, the original reward with non-boolean values (e.g. real numbers) may be too specific to be obtained again such that $M_{b,ij}$ would likely to be at most 1 for all i and j.

4. Sample from each bucket once initially, updating d and T according to the update algorithm described below. The purpose of this is to avoid division by zero when computing Q, as $T_j(t-1) = 0$ at initialization.
5. Initialize the time, t, to zero along with the allotted time for falsification. Start the time.

Sampling

1. Compute a matrix $\hat{\mu}$ of size $k \times N$ where $\hat{\mu}_{ij}$ approximates the observed reward from sampling the jth bucket for the ith feature by computing $\sum_b M_{b,ij}$.
2. Compute a matrix Q based on Eq. 3.1 above. For the confidence parameter, we use a time-dependent value of $\frac{1}{\delta} = t$ to reflect the increasing knowledge of buckets' rewards over time.
3. To sample a scenario, we sample a bucket for each feature. For each ith feature, select the jth bucket with the highest Q value by computing $j^* = \arg\max_j Q_{ij}$. *Break ties uniformly at random.* This is a key step in the sampling process as it is frequently the case initially that several buckets may have the exact same Q_j value. We break ties this way to avoid bias towards any specific bucket. Sample uniformly randomly within the domain that the bucket j^* represents.

Updating Internal State

1. Given the reward vector ρ, we compute our vector of booleans b as described in the **Setup** stage.
2. If b does not exist in the dictionary d and is among the set of worst boolean rewards found so far, i.e. $\forall b' \in d.\ b' \not\succ b$, then add b as a key to the dictionary d and initialize its value as $0^{k \times N}$.
3. For any $b' \in d$ such that $b \succ b'$, remove b' from d.
4. Increment the count M_b at each position $M_{b,ij}$ for the jth bucket sampled from the ith feature.
5. After the update, increment the time, t, by 1. If t is greater than the afforded time for falsification, abort. Otherwise, return to the **Sampling** stage.

[1]For example, if the ith property is that a self-driving car maintains a safety euclidean distance of 5 meters with respect to a pedestrian. Suppose ρ_i computes the negation of the distance. To convert this property's non-boolean value to a boolean, we may threshold such that the value becomes true if the distance is greater than -5 meters, otherwise, false.

3.2 Experiment

We conduct two experiments to evaluate (i) the time efficiency gained from parallelization; (ii) the effectiveness of the multi-armed bandit (MAB) sampler in balancing exploration and exploitation; and (iii) the effectiveness of VerifAI to falsify multiple objectives. We have developed a library of Scenic scripts based on the list of pre-crash scenarios described by the National Highway Traffic Safety Administration (NHTSA) [127]. These scripts cover a wide variety of common driving situations, such as driving through intersections, bypassing vehicles, and accounting for pedestrians. For our experiments, we selected 7 of these scenarios[2], running the VerifAI falsifier on each one in CARLA [44] for 30 minutes, with individual simulations limited to 300 time steps ($\sim$30 seconds). All parallelized experiments were run using 5 worker processes to perform simulation. We compare the MAB sampler with the existing passive and active samplers in VerifAI, which are Halton [74] and cross-entropy [148] samplers, respectively.

3.2.1 Time Efficiency of Parallelization

In our first experiment, we investigate the time efficiency gains from parallel falsification. For this experiment, we use a single objective specifying that the centers of the ego vehicle and other vehicles must stay at least 5 meters apart at all times. This specification means that counterexamples (i.e. scenarios in which the ego vehicle violates the objective) approximately correspond to collisions or near-collisions. Across the seven scenarios, we compare the number of simulations completed using serial versus parallel simulation with Halton, cross-entropy, an multi-armed bandit samplers. Also, we compare the number of counterexamples identified in each case.

The bar chart in Fig. 3.3 summarizes the results. For each scenario on the x-axis, the number of counterexamples (orange) and the number of total completed simulations are highlighted with different color bars. We observe that parallel simulations notably identify more counterexamples and completes more simulations than serial simulations. To quantify the observed time efficiency from parallel simulation, we use *speedup factor* metric, which is the ratio of the number of sampled scenarios in parallel over serial falsification. The speedup factor results in Table 3.1 show that we gain 3-5x speedup in the number of simulations using 5 parallel simulation processes. The variation in the number of samples generated can be attributed to *termination conditions* set in Scenic, which terminate simulations early if specific conditions are met. For some of these scenarios, termination occurred much sooner on average than other scenarios, leading to more simulations completing in 30 minutes. In regards counterexamples, we observe that cross-entropy sampler tends to find the most number of counterexamples, followed by MAB sampler and then the Halton sampler with the lowest performance.

[2]Refer to Appendix A for the descriptions of these 7 scenarios

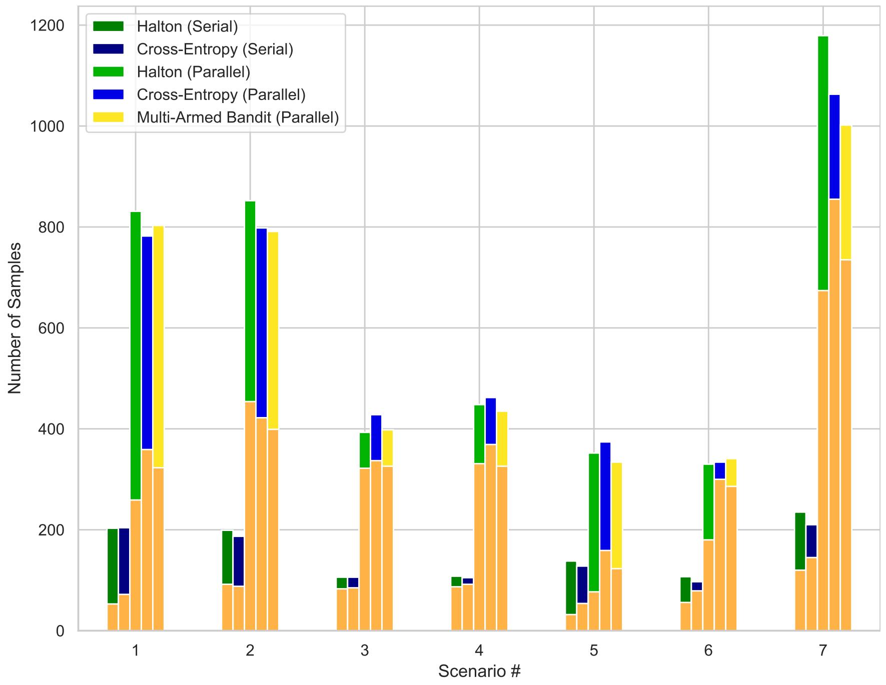


Figure 3.3: Comparison of (i) the serial and parallel versions of the falsifier for cross-entropy and Halton sampling and (ii) the multi-armed bandit sampler with the cross-entropy and Halton samplers all in parallel. The orange part of the bars represent the number of counterexamples found out of the total number of samples.

Scenario #	1	2	3	4	5	6	7
Speedup Factor	3.96	4.27	3.87	4.27	2.73	3.26	5.04

Table 3.1: The speedup factor and confidence interval width ratio metrics for the 7 scenarios.

3.2.2 Effectiveness of MAB Sampler: Exploration vs. Exploitation

In this section, we further analyze the results of the first experiment to validate whether the MAB sampler adequately balances the exploration and the exploitation of the scenario sample space. Fig. 3.4 summarizes the comparison by visualizing a scatter plot per sampler. The scattered dots in each plot represent the simulated scenarios per sampler. The colors of the dots represent the outcome of each simulation: orange (counterexample) and blue (safe example) based on the objective we use. The x, y, and z axes of the plot represent three different semantic features with continuous uniformly random distributions (i.e. intervals over real numbers). We observe noticeable patterns emerge from the scatter plots of the three samplers. The cross-entropy tend to exploit the smallest subset of the scenario sample space. Although cross-entropy sampler finds the highest number of counterexamples, this analysis reveals that the diversity of counterexamples as well the overall simulations are

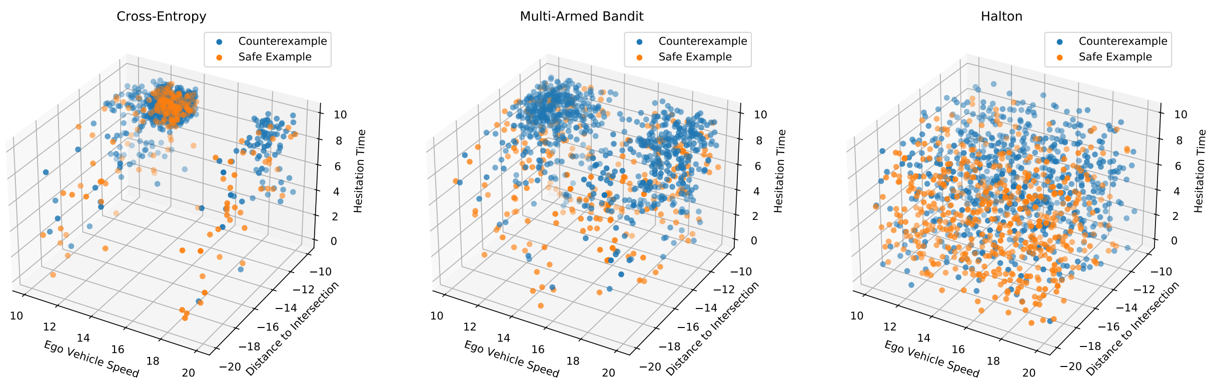


Figure 3.4: Comparison of points sampled for cross-entropy, MAB, and Halton samplers.

biased to specific subset of the sample space.

In contrast, Halton sampler explores the sampling space the most, but in trade-off performs the worst in finding counterexamples. Meanwhile, Fig. 3.3 visually shows that the MAB sampler balances the two extreme sampling patterns of the active sampler (cross-entropy) and the passive sampler (Halton). MAB sampler explores the sampling space better than the cross-entropy while less so than Halton. Yet, it exploits more than Halton to find more counterexamples. We observe these three patterns across the seven scenarios we experiment with.

3.2.3 Effectiveness of Falsifying Multi-Objective

For this experiment, we investigate the effectiveness of the MAB sampler in falsifying multiple objectives. We compare the falsification performances of the Halton, cross-entropy, and MAB samplers in serial versus parallel simulation cases, with respect to three distinct rulebooks with the same five properties but different preorder relations: (a) a completely *disconnected* graph representing no preference ordering, (b) a *linked list* structure $L \triangleq \rho_1 >_R \rho_2 >_R \ldots >_R \rho_5$ representing a total ordering, and (c) the graph G on the right in Fig. 3.2, which is the mixture of (a) and (b) in structure. We use a Scenic program that instantiates the ego vehicle, along with 5 adversarial vehicles at random positions with respect to a 4-way intersection and has all of them drive towards the intersection and either go straight or make a turn. The multi-objectives specify metric components ρ_j which say the ego vehicle must stay at least 5 meters away from vehicle j for all $j \in \{1, 2, 3, 4, 5\}$.

We find that when using L or G rulebooks, all three samplers falsify 4 of the 5 objectives with serial falsification, and all 5 objectives with parallel simulation in 30 minutes. When having no preference ordering, the three samplers falsify 3 of the 5 objectives with serial falsification, and 4 of the 5 objectives in the parallel case. As a case study, we have utilized the multi-objective falsification method in experiments with the LGSVL simulator [144]. Using a multi-objective specification with a variety of common driving situations, we were able to generate a wide range of test cases that cover much of the space of possible scenarios. These experiments were run with Apollo, an open-source autonomous driving software stack [10]. We discovered a number of bugs in Apollo using these new capabilities of VerifAI and Scenic [182]. For example, in multiple runs of scenarios involving intersections, Apollo gets "stuck" at stop signs and does not complete turns. We also noticed that Apollo sometimes stops far beyond the white entrance line at an intersection, potential hazardous scenarios in reality.

3.3 Bibliographic Notes

Previous literature individually address different aspects of falsification. There have been related work on falsification or, conversely, optimization of multiple objectives [27, 11, 12, 202, 135]. There are other tools that address simulation-based parallel testing of CPS [8, 133, 2]. However, these methods and tools do not provide support to model complex and reactive environments of autonomous CPS. We extend VerifAI as it provides an intuitive formalism, Scenic, to model and generate such environments. Building on to this benefit, we integrate prior techniques related to multi-objective falsification and parallel simulation to VerifAI for effective and scalable falsification.

3.4 Chapter Summary

The increasing autonomy of CPS has equipped these systems to operate in complex environments with stochastic, dynamic, and reactive physical agents. Although many simulation-based falsification tools have been proposed, they rarely provide adequate support to model and generate the operating environments of these autonomous systems in simulation. We extend VerifAI as it provides such support with Scenic. In particular, we enable VerifAI to (i) formally specify multi-objectives of a system using rulebooks, (ii) implement a multi-arm bandit sampler to effectively falsify a system, and (iii) parallelize simulations. Our experiment results show that these extensions could effectively scale the testing of AI/ML-based autonomous CPS in simulation. There are interesting directions for future work. In practice, it may not be straight forward how to order priorities over various properties of CPS. Hence, it may be interesting to investigate the effectiveness of randomizing the topological structure of rulebooks on falsification. Furthermore, it would be helpful to compare the strengths of other competing active and passive samplers.

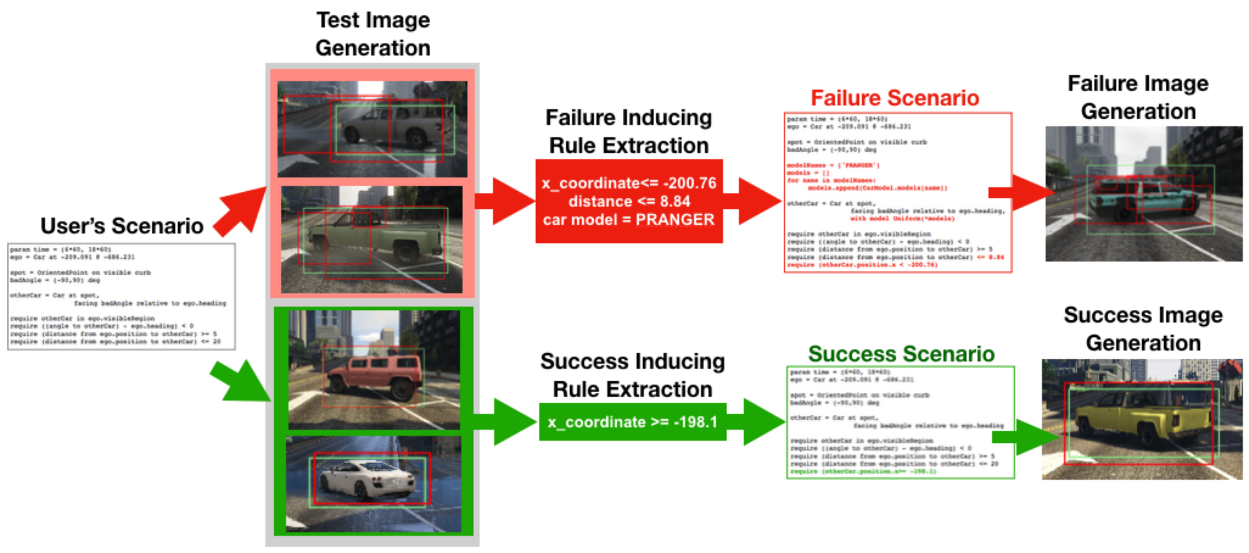
Chapter 4

Programmatic and Semantic System Debugging

In Chapter 3, we focus on searching for system-level failures. Yet, these failures do not explain *why* these failures occur. Hence, in this chapter, we focus on system-level debugging. Even as deep neural networks (DNN) have become very effective for tasks in perception, it remains difficult to explain and debug their behavior. We present a *programmatic* and *semantic* approach to explaining, understanding, and debugging the correct and incorrect behaviors of a neural network-based perception system [89]. Our approach is *semantic* in that it employs a high-level representation of the distribution of environment scenarios that the detector is intended to work on. It is *programmatic* in that a scenario representation is a program in a domain-specific probabilistic programming language which can be used to generate synthetic data to test a given perception module. Our framework assesses the performance of a perception module to identify correct and incorrect detections, extracts rules from those results that semantically characterize the correct and incorrect scenarios, and then specializes the probabilistic program with those rules in order to more precisely *describe* the scenarios in which the perception module operates correctly or not. We demonstrate our results using the Scenic probabilistic programming language and a neural network-based object detector. We demonstrate that our approach is effective, producing rules and refined programs that significantly increase the correct detection rate (from 65.3% to 89.4%) and incorrect detection rate (from 34.7% to 87.2%) of the network and can thus help with understanding, debugging and retraining the network.

4.1 Problem Statement

The key idea of our approach is to leverage the high-level semantic features formally encoded in a Scenic program to derive rules (sufficient conditions) that explain the behavior of a detection module in terms of those features. Our hypothesis is that since these features describe the important characteristics that should be present in an image and furthermore



User's Scenario

```
param time = (6*60, 18*60)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = (-90,90) deg

otherCar = Car at spot,
                facing badAngle relative to ego.heading

require otherCar in ego.visibleRegion
require ((angle to otherCar) - ego.heading) < 0
require (distance from ego.position to otherCar) >= 5
require (distance from ego.position to otherCar) <= 20
```

Failure Scenario

```
param time = (6*60, 18*60)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = (-90,90) deg

modelNames = ['PRANGER']
models = []
for name in modelNames:
     models.append(CarModel.models[name])

otherCar = Car at spot,
                facing badAngle relative to ego.heading,
                with model Uniform(*models)

require otherCar in ego.visibleRegion
require ((angle to otherCar) - ego.heading) < 0
require (distance from ego.position to otherCar) >= 5
require (distance from ego.position to otherCar) <= 8.84
require (otherCar.position.x < -200.76)
```

Success Scenario

```
param time = (6*60, 18*60)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = (-90,90) deg

otherCar = Car at spot,
                facing badAngle relative to ego.heading

require otherCar in ego.visibleRegion
require ((angle to otherCar) - ego.heading) < 0
require (distance from ego.position to otherCar) >= 5
require (distance from ego.position to otherCar) <= 20
require (otherCar.position.x>= -198.1)
```

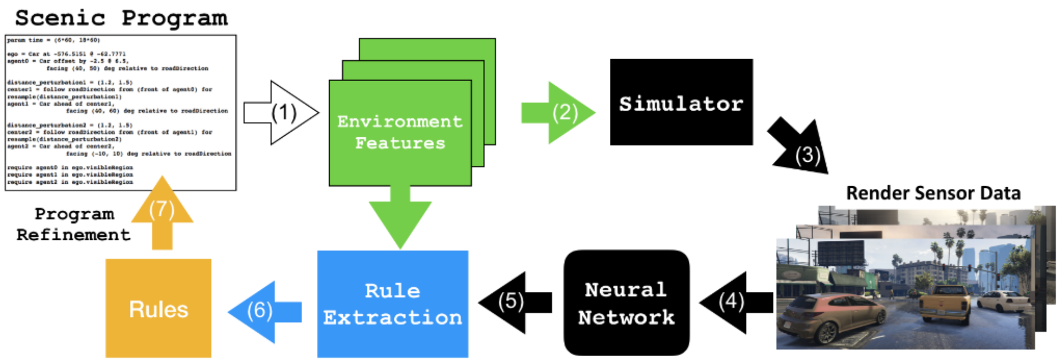
Figure 4.1: Overview of our approach. The green and red bounding boxes in the images are ground truth and prediction, respectively.

they are much fewer than the raw, low-level pixels, they should lead to small, compact rules that have a clear meaning for the developer.

The problem that our technique aims to address can be formalized as follows. Suppose a function g defines a sensor renderer in a simulator, which maps from a semantic feature vector, $[f_1, f_2, ..., f_n] \in D_1 \times D_2 \times ... \times D_n$, to synthetic sensor data $s \in S$, where each D_i represents the domain of the semantic feature f_i, and S is the domain of s. Let function m denote the given perception module. Finally, let e be an evaluation function which compares the perception module's prediction to the ground truths, and outputs a boolean class (correct or incorrect) based on a certain performance threshold. We assume that the necessary ground truth labels for the sensor data are collected in the simulator in use. Given a Scenic program, according to its feature dependencies and hard and soft constraints, the feature space, $D_1 \times D_2 \times ... \times D_n$, is defined. The problem is to find a subset feature space, $d_1 \times d_2 \times ... \times d_n \subseteq D_1 \times D_2 \times ... \times D_n$ such that when we sample a certain number of features $[f_1, f_2, ..., f_n] \in d_1 \times d_2 \times ... \times d_n$, the probability that $e(m(g([f_1, f_2, ..., f_n])))$ is equal to a target class (correct or incorrect) is *maximized*, while finding the most succinct rule that maximizes the probability. This trade-off is necessary to avoid the identified subset trivially containing a single element.

4.2 Methodology

A high-level overview of our analysis pipeline is illustrated in Figure 4.2. We start with a Scenic program that encodes constraints (and distributions) over high-level semantic features that are relevant for a particular application domain, in our case object detection



```
param time = (6*60, 18*60)

ego = Car at -576.5151 @ -62.7771
agent0 = Car offset by -2.5 @ 6.5,
           facing (40, 50) deg relative to roadDirection

distance_perturbation1 = (1.2, 1.5)
center1 = follow roadDirection from (front of agent0) for
resample(distance_perturbation1)
agent1 = Car ahead of center1,
                facing (40, 60) deg relative to roadDirection

distance_perturbation2 = (1.2, 1.5)
center2 = follow roadDirection from (front of agent1) for
resample(distance_perturbation2)
agent2 = Car ahead of center2,
                facing (-10, 10) deg relative to roadDirection

require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
require agent2 in ego.visibleRegion
```

Figure 4.2: Analysis Pipeline

for autonomous driving. Intuitively, the program (henceforth called scenario) encodes the environments that the user wants to focus on in order to test the module. Based on this scenario, we take step (1) as shown in Figure 4.2, where SCENIC samples a set of *semantic feature vectors* with concrete values (as highlighted in green boxes in Figure 4.2) by sampling from the specified distributions. An example of semantic features and their associated ranges of discrete or continuous values are shown in Table 4.1, and an example SCENIC program modeling a distribution of scenes using these semantic features are shown in Figure 4.3. A set of sampled concrete semantic feature vectors are inputted to a simulator to render a set of realistic, synthetic sensor data based on those semantic features, as shown in step (2) and (3) of the figure.

Next, as in step (4), the sensor data are inputted to the learning-based model under evaluation. Each sensor data is assigned a binary tag, correct or incorrect, based on the performance of the perception model on the sensor data. The tags obtained for the sensor data are mapped back to the semantic feature vectors, i.e. scenes, that led to the generation of the respective sensor data. The result is a labeled data set that maps each high-level semantic feature vector to the the respective tag. As shown in step (5), these labeled dataset and associated tags are inputted to rule extraction algorithms.

We then use off-the-shelf methods to extract *rules* from this data set as in step (6). The rule extraction is described in more detail in Sec. 4.2.1. The result is a set of rules encoding the conditions on high-level features that lead to likely correct or incorrect performance. As in step (7), these rules are then encoded into the original SCENIC program to further refine its distribution.

4.2.1 Rule Extraction

Methods: In principle, any off-the-shelf classification algorithm can be used. We employ and compare two different methods, decision tree (DT) learning for classification [134] and anchors [139], to extract rules capturing the subspace of the semantic feature space defined in the given SCENIC program. Decision tree learning is commonly used to extract rules explaining the *global* behavior of a complex system, while the anchors method is a state-of-

Feature	Range
Weather	Neutral, Clear, Extrasunny, Smog, Clouds, Overcast, Rain, Thunder, Clearing, Xmas, Foggy,Snowlight, Blizzard, Snow
Time	[00:00, 24:00)
Car Model	Blista, Bus, Ninef, Asea, Baller, Bison, Buffalo, Bobcatxl, Dominator, Granger, Jackal, Oracle, Patriot, Pranger
Car Color	R = [0, 255], G = [0, 255], B =[0, 255]
Car Heading	[0, 360) deg
Car Position	Anywhere on a road on GTA-V's map

Table 4.1: Environment features and their ranges in GTA-V

the art technique for extracting explanation rules that are *locally* faithful.

A decision tree intuitively encodes decisions in a tree-like structure, starting from a root node branching out to subsequent nodes below according to different conditions. They are highly interpretable, provided that the trees are short. One can easily extract rules for explaining different outcomes, by simply following the paths through the trees and conjuncting the decisions encoded in the tree nodes.

The anchor method is a state-of-the art technique, back in 2019, that aims to explain the behavior of complex ML models with high-precision rules called anchors, representing local, sufficient conditions for predictions. The system can efficiently compute these explanations for any black-box model with high-probability guarantees. We extract anchors over the high-level semantic features.

Blackbox vs Whitebox Analysis: So far we explain how we can obtain rules when treating the detection module as a black box. We also investigate a white-box analysis, to determine whether we can exploit the information about the internal workings of the module to improve the rule inference. The white-box analysis is one of our novel contributions. We leverage recent work [67] which aims to infer likely properties of neural networks. The properties are in terms of on/off activation patterns at different internal layers that lead to the same predictions. These invariant patterns are computed by applying decision-tree learning over the activations observed during the execution of the neural network on a training or test dataset. For example, an invariant pattern, if triggered, always result in incorrect detection with respect to the dataset.

However, the support of the invariant activation patterns may be low. In other words, it is possible that these invariant activation patterns may represent the given perception model's responses to a small proportion of the overall incorrectly detected sensor data in the provided dataset. Yet, we hypothesize that these invariant patterns capture more focused properties (or rules) among sensor belonging to a target class. Hence, using these invariant patterns, we augment the binary (e.g. correct vs incorrect detection) target class to provide

```
param time = Range(6, 18)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = Range(-90, 90) deg
otherCar = Car at spot,
              facing badAngle relative to ego.heading

require otherCar in ego.visibleRegion
require 5 <= (distance from ego to otherCar) <= 20
```

Figure 4.3: An example Scenic program modeling Scenario 1 (refer to Sec. 4.3.1)

more signals for the rule extraction algorithms to better identify rules, in the following way. For example, using an invariant pattern for the correct class, we created two sub-classes for the existing "correct" target class. By feeding in only sensor that are previously correctly detected by the perception module, the sensor data that trigger the invariant pattern is re-labelled as "correct-by-invariant-pattern," otherwise, "correct-unlabelled." Likewise, we apply the same augmentation for the "incorrect target" class.

Rule Selection Criteria: Once we extract rules with either DT or anchors, we select the best rule using following criteria. To best achieve our objective, first, we choose the rule with highest precision on a held-out testset of feature vectors. If there are more than one rule with equal high precision, then we choose the rule with the highest coverage (i.e. the number of feature vectors satisfying the rules). Finally, if there is still more than one rule left, then we break the tie by choosing the most compact rule which has the least number of features. The last two criteria are established to select the most general high-precision rule.

In this section we report on our experiments with the approach on the deep neural network-based object detector. We investigate whether we can synthesize rules that are effective in generating test inputs that increase the probability of correct/incorrect detection, thus explaining the correct/incorrect behavior of the analyzed module. We evaluate the techniques along the following dimensions: decision tree (DT) vs anchor, black-box (BB) vs white-box (WB).

4.3 Experiment

4.3.1 Study Design

We use a pre-trained SqueezeDet [194] object detector model that is already trained on 10,000 RGB images rendered in GTA-V game simulator [63], involving one to four cars in various locations of the map with diverse backgrounds. The perception task is to detect vehicles on road with two-dimensional bounding boxes.

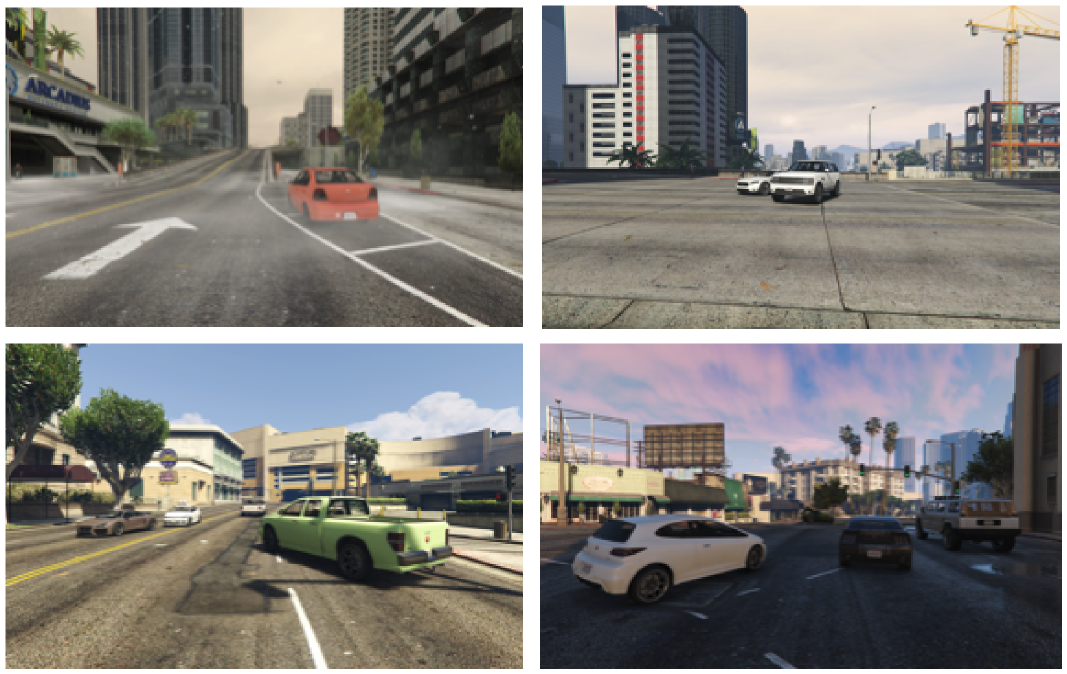


Figure 4.4: From top-left one-car image, each image corresponds to scenario 1, 2, 3, and 4 in a clockwise manner. The scenario number is the number of cars

Training and Test Sensor Data Generation with Scenic Programs

Using our domain knowledge of driving, we modeled four different distributions of scenarios as four Scenic programs[1]. The scenes, i.e. concrete values of environment semantic features, are sampled from the the programs. These scenes are rendered as RGB images by the GTA-V simulator, along with the corresponding ground truth two-dimensional bounding boxes. Sensor data generated from these scenarios are shown in Figure 4.4. Using each Scenic program, we generated 950 images as a train set and another 950 new images as a test set.

1. Scenario 1 describes the situation where a car is illegally intruding over a white striped traffic island at the entrance of an elevated highway. A Scenic program modeling Scenario 1 is shown in Fig. 4.3.
2. Scenario 2 describes two-car scenario where one car occludes the ego car's view of another car at a T-junction intersection on an elevated road.
3. Scenario 3 describes scenes where other cars are merging into ego car's lane. The location in this scenario is carefully chosen such that the sun rises in front of ego car, causing a glare.
4. Scenario 4 describes a set of scenes when nearest car is abruptly switching into ego car's lane while another car on the opposite traffic direction lane is slightly intruding over the middle yellow line into ego car's lane.

[1]Please refer to Appendix B for Scenic programs of scenario 2,3, and 4.

Evaluation Metric

We assign binary target class to each RGB image based on the object detector's correctness, which we define using the F1 score metric (harmonic mean of the precision and recall) [101]. This metric is commonly used in statistical analysis of binary classification. The F1 score is computed in the following way. For each image, the true positive (TP) is the number of ground truth bounding boxes correctly predicted by the detection module. Correctly predicted here means intersection-over-union (IoU for object detection) is greater than 0.5. The false positive (FP) is the number of predicted bounding boxes that falsely predicted ground truths. This false prediction includes duplicate predictions on one ground truth box. The false negative (FN) is the number of ground truth boxes that is not detected correctly. We compute the F1 score for each image, and if it is greater than a threshold, we assigned *correct* label; if not, *incorrect*. The threshold used in our experiments was 0.8.

Blackbox and Whitebox Setup

We use off-the-shelf algorithms for rule extraction. For decision tree classification algorithm, we use the *rpart* [170] package in R software, which implements corresponding algorithm in [19], with default parameters. Also, we use the anchor algorithm with its code from [138] with the default parameters.

We analyze the architecture of the SqueezeDet network and determine that there are three maxpool layers which provide a natural decomposition of the network. Furthermore, these layers have relatively low dimensionality making them good targets for property inference. We consider activation patterns over maxpool neurons based on whether the neuron output is greater or equal to zero. A decision tree can then be learned over these patterns to fit the prediction labels. For our experiments we select patterns from the maxpool layer 5, which turn out to be highly correlated to images that lead to correct/incorrect predictions. Using the training dataset, we augment the target class corresponding to the Squeeze maxpool layer 5 decision pattern as p5c(correct) and p5_ic(incorrect) and the remaining as correct_unlabelled and incorrect_unlabelled, respectively.

From the train set, we extracted rules to predict each target class based on the feature vectors. These rules were evaluated on the test set based on precision, recall, and F1 score metrics. For DT learning we adjusted the label weight to account for the uneven ratio among labels for both black-box and white-box labels. For the anchor method, we applied it on each instance of the training set until we had covered a maximum of 50 instances for every label (correct, incorrect for Black Box, and p5c, p5_ic, correct_unlabelled, incorrect_unlabelled for White Box). The best anchor rule for every label is selected based on the rule selection criteria mentioned in section 4.2.1.

Semantic Feature Augmentation

We augmented the feature vector with some extra features that are not part of the feature values provided by the simulator but are used in SCENIC programs. The reason is that the

Scenario # (Baseline→Rule Precision)	Correct Detection Inducing Rules
Scenario 1 (65.3% → 89.4%)	**x coordinate** $\geqslant$ -198.1
Scenario 2 (72.3% → 82.3%)	**hour** $\geqslant$ 7.5 $\wedge$ **weather** = all except neutral $\wedge$ **car0 distance from ego** $\geqslant$ 11.3m $\wedge$ **car0 model** = {Asea, Bison, Blista, Buffalo, Dominator, Jackal, Ninef, Oracle}
Scenario 3 (61.7% → 79.4%)	**car0 red color** $\geqslant$ 74.5 $\wedge$ **car0 heading** $\geqslant$ 220.3 deg
Scenario 4 (89.6% → 96.2%)	**car0 model** = {Asea, Baller, Blista, Buffal, Dominator, Jackal, Ninef, Oracle}

Table 4.2: The identified rules that induce correct detection for the four scenarios explained in Sec. 4.3.1, respectively. These results summarize the best outcomes shown in Table 4.6.

set of feature values provided by the simulator may be too sparse to find any rules among them. Also, Scenic can utilize the *relations* among the features, whose values are not provided by the simulator. For example, in Scenario 1 (shown in Fig. 4.3), the *distance* from ego to otherCar is not part of the feature values provided by GTA-V but used in line 10 of the program. However, it can be computed with Euclidean distance metric using (x,y) location coordinates of ego and otherCar. Also, the *difference in heading angle* between ego and otherCar is also added as extra feature to represent `badAngle` variable in line 7 of the program. We find this augmentation to be helpful. For instance, in Table 4.4, for Scenario 1, the failure inducing rule with the highest precision includes the *distance* feature which we augmented.

4.3.2 Results

Tables 4.2 and 4.4 show the best rules (wrt. precision) extracted with our framework, along with the baseline correct/incorrect detection rate for each given scenario and the detection rate for the generated rules. This precision is computed using our test dataset. Hence, of the 950 concrete semantic feature vectors per Scenic program for testing (refer to Sec. 4.3.1), we retrieve the vectors that satisfy the identified rule. The *Rule Precision* in Table 4.2 and 4.4 is defined as the proportion (in percentage) of the retrieved vectors that result in either correct or incorrect detection. On the other hand, *Baseline* in Table 4.2 and 4.4 represents the proportion of overall 950 vectors per Scenic program which result in correct or incorrect detection. The results indicate that indeed our framework can generate rules that significantly increase the correct and incorrect detection rate of the module. Furthermore, the generated rules are compact and interpretable.

Scenario #	1	2	3	4
Original Program	0.347	0.277	0.383	0.104
BB Decision Tree	0.703	0.418	0.506	0.375
WB Decision Tree	0.73	0.449	0.494	0.099
BB Anchor	0.872	0.357	0.834	0.573
WB Anchor	0.674	0.422	0.365	0.176

Table 4.3: The precision of incorrect detection inducing rules over 500 new sensor data generated from refined Scenic programs, respectively. The justification for specifically using 500 sensor data is provided in Figure 4.6.

Our programmatic approach which outputs correct (or incorrect) performance inducing rules using interpretable semantic features enable intuitive debugging of a learning-based model, *without having to manually look through a large corpus of sensor data*. For example, in Scenario 1, from the rules found using our approach, we identify a counter-intuitive characteristic of the pre-trained perception model used in this study. As shown in Table 4.2, the correct detection inducing rule for Scenario 1 is "x coordinate $\geqslant -198.1$." In GTA-V, given that ego car's location is fixed, the condition on the `otherCar`'s x-coordinate is equivalent to the `otherCar`'s distance from ego being greater than 11m. Note that the RGB camera is mounted on the ego car, and the `otherCar` is defined in the Scenic program of Scenario 1 in Figure 4.3. On the other hand, as shown in Table 4.4, the incorrect detection inducing rule for Scenario 1 states that the `otherCar` to be within 8.84m and its car model to be PRANGER. In short, the pre-trained model actually performs worse when the other vehicle of a specific model is closer, and better when further away. We validate this finding by testing the pre-trained model on such scenes, which are visualized in Figure 4.5.

Results for Correct Behavior: Tables 4.5 and 4.6 summarize the results for the rules explaining correct behavior. The results indicate that there are clear signals in the heavily abstracted feature space and they can be used effectively for scenario characterization via the generated high-precision rules.

Also, the results show that DT learning extracts rules with better F1 scores for all scenarios as compared to anchors. This could be attributed to the difference in the nature of the techniques. The anchor approach aims to construct rules that have high precision in the locality of a given instance. Decision-trees on the other hand aim to construct global rules that discriminate one label from another. Given that a large proportion of instances are detected correctly by the analyzed module, the decision tree is able to build rules with high precision and coverage for correct behavior.

The results also highlight the benefit of using white-box information to extract rules for correct behavior. Table 4.7 shows the support for the decision pattern is significant (greater than 65% on average for all scenarios). The support is defined as a correlation of

Scenario # (Baseline→Rule Precision)	Failure Detection Inducing Rules
Scenario 1 (34.7% → 87.2%)	**x coordinate** $\leqslant$ -200.76 $\wedge$ **distance** $\leqslant$ 8.84 $\wedge$ **car model** $=$ PRANGER
Scenario 2 (27.7% → 44.9%)	**hour** $\geqslant$ 7.5 $\wedge$ **weather** $=$ all except Neutral $\wedge$ **car0 distance from ego** $<$ 11.3
Scenario 3 (38.3% → 83.4%)	**weather** $=$ neutral $\wedge$ **agent0 heading** $= \leqslant$ 218.08 deg $\wedge$ **hour** $\leqslant$ 8.00 $\wedge$ **car2 red color** $\leqslant$ 95.00
Scenario 4 (10.4% → 57.3%)	**car0 model** $=$ PATRIOT $\wedge$ **car1 model** $=$ NINEF $\wedge$ **car2 model** $=$ BALLER $\wedge$ 92.25 $<$ **car0 green color** $\leqslant$ 158 $\wedge$ **car0 blue color** $\leqslant$ 84.25 $\wedge$ 178.00 $<$ **car2 red color** $\leqslant$ 224

Table 4.4: Rules for incorrect behaviors of detection module with the highest incorrect precision from Table 4.3

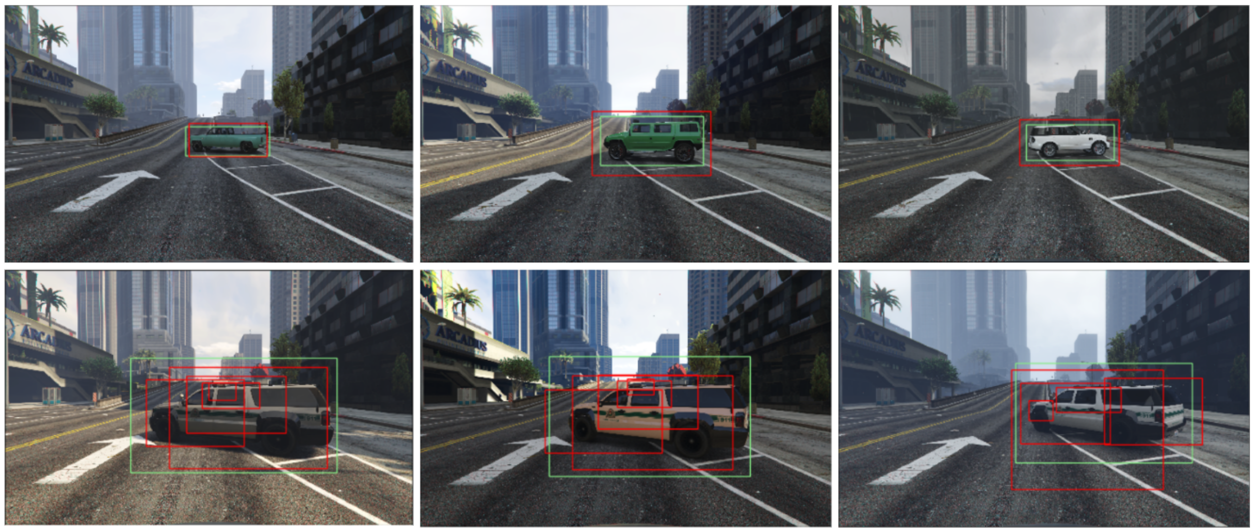


Figure 4.5: The top (or bottom) three images are generated using the refined SCENIC program with the correct (or incorrect) detection inducing rule shown in Table 4.2 (or Table 4.4) for Scenario 1. The green boxes are the ground truth bounding boxes for the detection task, and red boxes are the predictions of the pre-trained perception model used in this study.

Scenario #	1	2	3	4
BB Decision Tree	0.723	0.342	0.631	0.622
WB Decision Tree	0.727	0.696	0.601	0.778
BB Anchor	0.361	0.457	0.302	0.438
WB Anchor	0.520	0.188	0.149	0.438

Table 4.5: F1 score of correct rules on testset

Scenario #	1	2	3	4
Original Program	0.653	0.723	0.617	0.896
BB Decision Tree	0.843	0.778	0.787	0.950
WB Decision Tree	0.826	0.823	0.788	0.962
BB Anchor	0.727	0.811	0.652	0.928
WB Anchor	0.894	0.817	0.794	0.928

Table 4.6: Precision of correct rules on the testset

the decision pattern to a specific label. Using this information to augment the labels of the dataset helped to improve the precision and F1 score of the rules (w.r.t. Scenic features) for both DT learning and anchor method.

Results for Incorrect Behavior: Tables 4.3 and 4.4 summarize the results for the rules explaining incorrect behavior. Rule derivation for incorrect behavior is more challenging than for correct behavior due to the low percentage of inputs that lead to the incorrect detection for a well trained network. In fact the F1 scores (computed on the test set) for rules predicting incorrect behavior are too low due to very low (in some cases 0) recall values.

To properly validate the efficacy of the generated rules, we refine the Scenic programs by encoding the rules as constraints and generate 500 new images. We then evaluated our module's performance on these new datasets. Figure 4.6 justifies our choice of 500 as the number of new images that we generate for evaluation.

All four methods contribute to more precisely identifying the subset features spaces in which the module performs worse. Specifically, Table 4.3 illustrates that the black-box anchor method enhances the generation rate of incorrectly detected images by 48% on average in Scenarios 1, 3, and 4 compare to the baseline. This is a significant increase in the ratio of incorrectly labelled images generated from the program, providing evidence that the refined programs are more precisely characterizing the failure scenarios.

We also note that the anchor method outperforms DT learning. This is expected, because the anchor method extracts rules that are highly precise within a local feature space. The exception is Scenario 2. We conjecture that the reason that the anchor method does not perform better than DT learning is due to uncontrollable non-determinism in GTA-V, which generate pedestrians in close vicinity to the camera of ego car even though its

Scenario #	1	2	3	4
Correct DP	0.626	0.651	0.514	0.824
Incorrect DP	0.276	0.175	0.234	0.212

Table 4.7: Support for correct and incorrect decision patterns

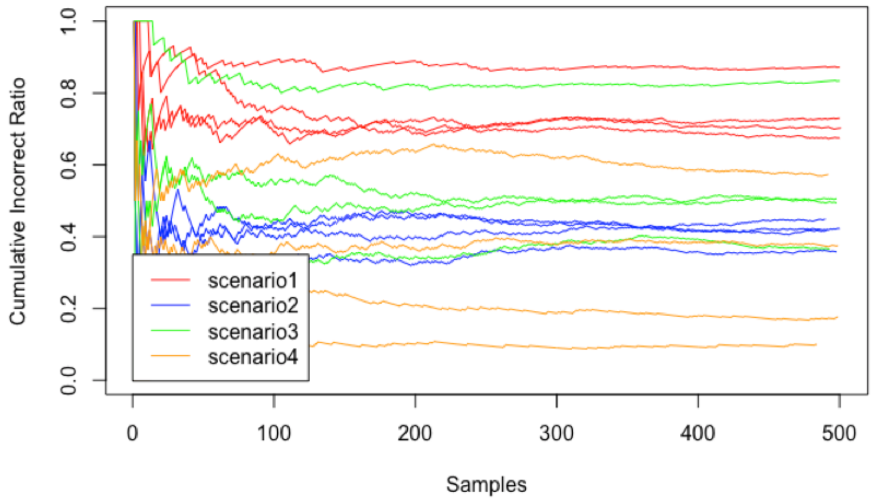


Figure 4.6: The cumulative ratio of incorrectly detected images generated from refined Scenic programs (using incorrect rules) stabilizes over 500 samples. Each color has four graphs representing four different rule extraction methods

Scenic program did not have any pedestrian. GTA-V non-deterministically instantiates these pedestrians, and this changes in the background may have affected perception. This is an issue with the GTA-V which originally is not built for data generation purpose. GTA-V does not allow users to control or eliminate these pedestrians and it does not provide features related to pedestrians during data collection process. In future work, we plan to incorporate simulators that allows a deterministic control (such as CARLA [44]) for further experimentation.

Unlike the results for correct behavior, the whitebox approach tends to perform worse than blackbox when focusing on incorrect behavior. This outcome can be attributed to very low support for decision patterns computed for incorrect behavior, with maximum of 27.6% among the four scenarios as shown in Table 4.7. However, we do observe that the white-box approach for both DT learning and anchors does, in general, enhance the ratio of incorrectly detected images as shown in Table 4.3, compared to those of the original programs.

Limitations: Our technique relies on abstracting an image with a high resolution (for instance 1920 x 1200 in our example) to a vector of a small set of semantic features. In our experiments we are able to derive compact rules with high precision and coverage. However, we do note that in other application domains, other than autonomous driving, the abstraction may lead to under-determined representation, which may not yield any noticeable patterns. Therefore, appropriate selection of a subset of essential features for a given application domain (facilitated by an appropriate definition using Scenic), is essential.

We also note that all the Scenic programs we experiment with contain only uniform distributions. Also, for each of the scenario programs that we analyzed, we fix the location and heading angle of the camera. We impose such restriction to prevent too much variations, or noises, from the signals to extract rules. Our methodology can apply to more general setting with variations in camera locations and angles. However, increase in variations may require higher number of images to better reflect the joint distribution of all dimensions of variations, from which rules can be adequately extracted.

4.4 Bibliographic Notes

Most techniques that aim to provide explainability and interpretability for deep neural networks (DNNs) in the field of computer vision focus on attributing the network's decisions to portions of the input images([113, 132, 158, 164, 201]). GradCAM [153] is a popular approach for interpreting CNN models that visualizes how parts of the image affect the neural network's output by looking into class activation maps (CAM). Other techniques focus on understanding the internal layers by visualizing their activation patterns [26]. Our approach, on the other hand, aims to provide characterizations at a higher level than raw image pixels, namely at the level of abstract features defined in a Scenic program.

Recent work aims to explain the decisions of DNNs in terms of higher-level concepts. The technique in [88] introduces the idea of concept activation vectors, which provide an interpretation of a neural network's internal state in terms of human-friendly concepts. Feature Guided Exploration [188] aims to analyze the robustness of networks used in computer vision applications by applying perturbations over high-level input features extracted from raw images. They use object detection techniques (such as SIFT – Scale Invariant Feature Transform) to extract the features from an image. In contrast to these techniques we directly leverage Scenic which defines the high-level features in a way that is already understandable for humans. Existing approaches typically use classification networks whose output directly corresponds to the decision being made and rely on the derivative of the output with respect to the input to calculate importance. In our application, there is no direct correlation between the output of the object detector network and the validity of the bounding boxes. Furthermore, unlike all previous work, we can use the synthesized rules to automatically generate more input instances, by refining the original Scenic program and then using it to generate data. These instances can be used to test, debug and retrain the network.

4.5 Chapter Summary

We present a semantic and programmatic algorithm for characterizing success and failure scenarios of a provided DNN-based perception model in the form of probabilistic programs. We use Scenic to represent operating environments of a system using semantic features. Also, we utilize Scenic as a generative model to generate sensor data with a simulator. The combination of Scenic's semantic representation and its generative capability allows us to associate high-dimensional sensor data with low-dimensional vectors of concrete semantic feature values. We leverage this benefit to abstract sensor data as semantic feature vectors and then analyze which set of features are highly correlated with correct or incorrect behaviors of the perception model. Our analysis identifies conditions, or rules, over the features. These rules are used to refine the input Scenic program to characterize success and failure scenarios of the perception model. Our experiment results show that the identified rules effectively identifies conditions over a set of semantic features which highly correlate to either the perception's correct or incorrect behaviors. Furthermore, our results demonstrate that with the refined Scenic programs we can systematically generate sensor data which likely induce the perception model to succeed or fail. These data can be potentially useful to systematically augment training dataset for fine-tuning. For future work, we plan on extending this approach to dynamic scenarios involving interactive behaviors among the agents in a scenario.

Part II

Sim-to-Real Validation of System Performance

Chapter 5

Formal Scenario-Based Track Testing

Part I focuses on identifying system failures and understanding why those failures occur at component level in simulation using synthetic sensor data. Part II is motivated by a skepticism towards the identified failures in simulation. Specifically, how can we validate that the component or system failures identified in simulation also transfer to reality? The reason for this validation is that there are known discrepancies between synthetic versus real sensor data. Some of these differences can be perceivable to human eyes when comparing real and synthetic RGB images. Meanwhile, other discrepancies are subtle like the ways how rays from LiDAR or radar reflect from various textures and shapes of physical objects in simulation versus reality. These discrepancies could result in different behaviors of NN-based models as shown in Fig. 5.1. This is not a conjecture, but a valid concern which led to the emergence of a new field called domain adaptation [184] within computer vision.

Track testing is a typical approach to test autonomous system in reality. It is an intermediate step between testing in simulation and on public roads. This form of testing has much

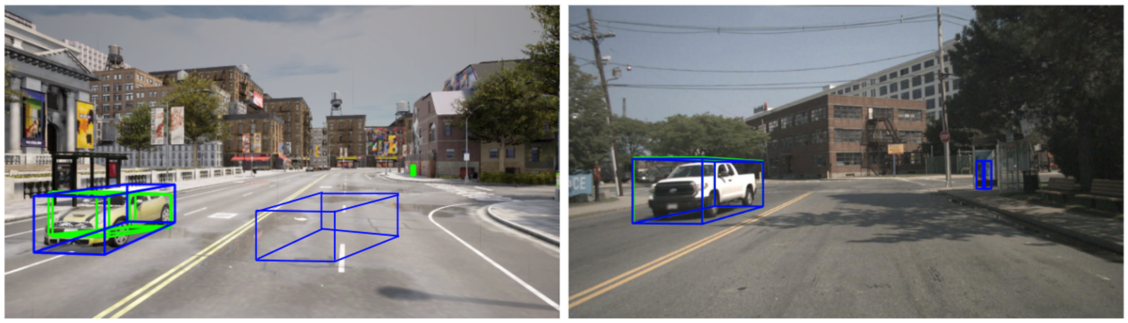


Figure 5.1: The discrepancy in the behaviors of a neural network-based perception model on the synthetic (left) versus the real (right) RGB camera images are shown. The contents of the two images are similar, where a single vehicle approaches from the opposite traffic. However, the perception on the synthetic image results in a false detection of a non-existent vehicle. The green boxes represent the ground truths, and the blue represent detections by the neural network.

lower risk of injury than testing on public roads. It involves a reasonable degree of control over the other agents around the AV, such as pedestrian dummies and inflatable cars to use for crash testing.

However, physically reconstructing these conditions, or scenarios, for track testing can be very expensive, labor-intensive, and time-consuming. Thus, it is necessary to carefully select test scenarios for track testing, which will prove most effective at identifying failures or strange behavior, uncovering bugs, and increasing assurance in the safety of the AVs. Hence, in this chapter, we take step towards addressing the following two questions:

1. *Can formal simulation aid in designing effective road tests for AVs?* By *formal simulation* we mean simulation-based testing that is guided by the use of formal models of test scenarios and formal specification of safety properties and metrics. More specifically, do unsafe (safe) runs in simulation produce unsafe (safe) runs on the track How should one select tests from simulation to run on the track?
2. *How well can simulation match track testing of AVs?* We aim to quantitatively and qualitatively compare simulation and track testing data, for a test scenario that has been formally specified and implemented in both simulation and track testing.

We conduct a study at track testing site with a real self-driving car, which operate with the same autopilot as tested in simulation with a replica of the map of the track in reality [61]. From extensive simulations, we select test cases for track testing regarding a distribution of scenarios modeling a pedestrian crossing and the self-driving having to yield to the pedestrian. Our results indicate that our formal simulation-based approach is effective at synthesizing test cases that transfer well to the track: 62.5% of unsafe simulated test cases resulted in unsafe behavior on the track, including a collision; 93.3% of safe simulated test cases resulted in safe behavior on the track (and no collisions). Our results also shed light on potential causes for AV failure in perception, prediction, and planning. While AV and pedestrian trajectories obtained in simulation and real-world testing for the same test were qualitatively similar (e.g., see Fig 5.6), we also noted differences as quantified using time-series metrics [65, 40] and with metrics such as minimum distance between the AV and pedestrian. Variations exist even amongst simulations of the same test case due to non-deterministic behavior of the AV stack, although these are smaller.

5.1 Problem Statement

Let $\mathcal{M}$ be the *simulation model*, including the full software stack and vehicle dynamics model of the AV, and its environment, including models of all other objects and agents in the simulated world. This model can be configured through a vector of *parameters* $\vec{\pi}$, typically supplied through a configuration file or suitable API to the simulator. Each valuation of $\vec{\pi}$ defines a *test case* x. We assume, for this section, that each test case x produces a unique

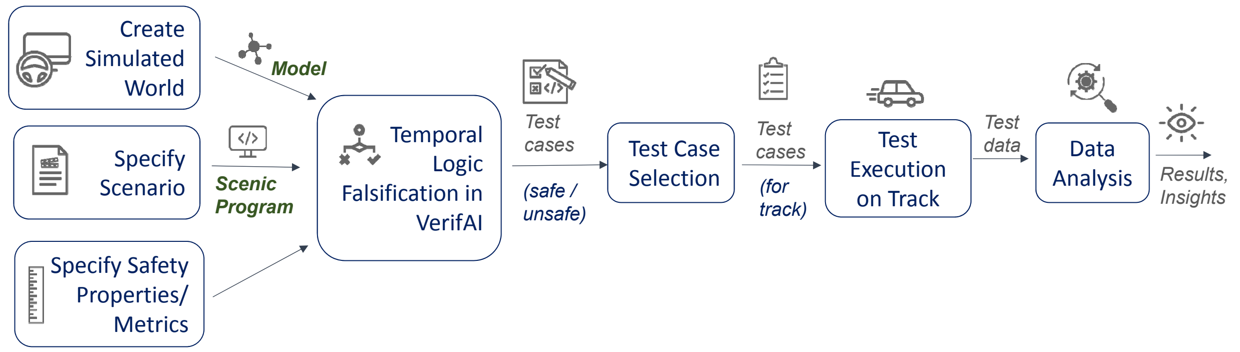


Figure 5.2: Overview of our approach to synthesize failure inducing test cases for track testing

simulation run.[1] The time series data generated by the simulation run is referred to as a *trace* τ. Each test case x is designed so as to also be implementable on the real AV on the track, although such implementation can be non-trivial as we describe later. On the track, the environment is less controllable than in the simulator, and therefore a single test case x_i can produce multiple test runs $r_{i,1}, r_{i,2}, \ldots$.

A key aspect of our method is to formally specify a set of test cases along with an associated probability distribution over them. We refer to this distribution of test cases as a *scenario* $\mathcal{S}$, which is defined by a Scenic program $P_{\mathcal{S}}$. Typically, a subset of simulation parameters $\vec{\pi}$ are modeled in $P_{\mathcal{S}}$, while the others are left fixed for the experiment.

5.2 Methodology

Our overall methodology is depicted in Fig. 5.2, and involves the following steps.
Create Simulation Model: First, the initial step involves generating a simulation environment that is photorealistic and incorporates dynamical models for various agents that can be utilized on the test track. This process includes creating high-definition (HD) maps, gathering sensor data, employing the collected data to develop a detailed three-dimensional (3D) mesh, uploading the mesh into the simulator, noting specific details of drivable areas in the simulator, and combining the resulting 3D world model with vehicle and agent dynamics models.
Formalize Test Scenario: The next step is to formalize the test scenario(s) to execute on the track. We take a formal methods approach, specifying test scenarios in Scenic.
Formalize Safety Property/Metric: In addition to the formal specification of a scenario, it is also necessary to define one or more properties that accurately reflect the conditions in which an autonomous vehicle (AV) can be considered to be operating safely. In the context of

[1] Note, however, that some industrial simulators and AV stacks tend to be non-deterministic in that the configurable parameters may not define a unique simulation run. We will discuss later the impact of such non-determinism on our results.

formal methods, safety properties that pertain to traces are typically expressed using logical notation, such as temporal logics. If these properties have a quantitative aspect, they are referred to as safety metrics.

Identify Safe/Unsafe Test Cases: Once the above three steps are complete, the simulation model, test scenario, and safety properties are fed into the VerifAI tool to perform falsification. The Scenic scenario P_S defines a distribution over the parameters $\vec{\pi}$. We configured VerifAI to sample from this distribution, simulating each corresponding test case and monitoring the safety properties on the resulting trace. VerifAI stores the sampled values of $\vec{\pi}$ in *safe* or *error* tables depending on whether the test satisfies or violates the specification. Moreover, VerifAI uses the *robust semantics* of metric temporal logic (MTL) [48] to compute a quantitative *satisfaction value* ρ for the specification φ which indicates *how strongly* it is satisfied: $\rho > 0$ implies φ is satisfied, and larger values of ρ mean that larger modifications to the trace would be necessary for φ to instead be falsified. The resulting test cases are fed to the next step.

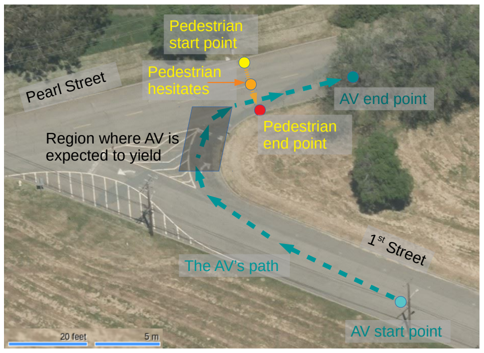
Select Test Cases for Track Testing: VerifAI offers various methods, including Principal Component Analysis and clustering, to automatically examine the safe and error tables and identify patterns. When working with low-dimensional spaces, direct visualization can also be utilized to pinpoint clusters of safe or unsafe tests. By employing either of these methods, diverse modes of behavior can be identified, and representative test cases can be selected for execution on the track.

Implement Selected Test Cases on Track: After identifying test cases in simulation, the next step is to execute them on the track. This process requires controlling dynamic agents, such as environment vehicles, pedestrians, and bicyclists, using parameters that are specified in the Scenic program and are synthesized into a test case, such as the starting location, time to begin motion, velocities, and other relevant factors. In addition, it is important to ensure that the hardware used for track testing can accurately replicate the synthesized tests in simulation, taking into account any limitations that the hardware may have, even if it is state-of-the-art.

Record Results and Perform Data Analysis: At the final stage of the process, during track testing, we collect several types of data such as videos of the AV in motion, data on the AV which includes sensor and log data from the AV software stack, and information from the test track hardware like GPS beacons and the hardware used to control dynamic agents such as a pedestrian dummy. Afterward, this data is analyzed to assess the efficacy of the selected test cases obtained through formal simulation, to compare the simulation traces with those obtained from track experiments, and to determine potential reasons behind any unsafe or intriguing behaviors exhibited by the AV.

5.3 Experiment

We conducted a case study at a track testing site at GoMentum [159] which is AAA's track testing site located in Concord, California.



```
ego = EgoCar at 38.6 @ 183.9,  # LGSVL coordinates
      facing 10 deg relative to roadDirection,
      with behavior DriveTo(40 @ 225.2)
ped = Pedestrian at 19.782 @ 225.680,
      facing 90 deg relative to roadDirection,
      with behavior Hesitate,
      with startDelay (7, 15),    # (a,b) = uniform
      with walkDistance (4, 7),   # distribution on
      with hesitateTime (1, 3)    # that interval
```

Figure 5.3: A bird's-eye view of the scenario S (left) and the (simplified) Scenic program P_S encoding our test scenario.

Simulation Model Creation: First, we created a digital simulation environment that closely mimics the real-world testing area for autonomous vehicles (AVs) called "Urban A" at Go-Mentum. The creation process involved collecting various data while driving around the site, including LiDAR point cloud data, camera images, and location data. The collected data was then processed and converted into a 3D mesh that represents every detail of the road surface and surrounding objects such as buildings, curbs, sidewalks, signs, and more. Textures were applied to the mesh using tens of thousands of captured images. Details of the drivable areas, including lane lines, driving directions, road speeds, crosswalks, intersections, and traffic signs, were annotated in the LGSVL Simulator [144]. The annotated, textured mesh was then compiled into a loadable simulation environment along with HD maps that were used both in simulation and in the AV for real-world testing. This process created a photorealistic simulation environment, which is a digital replica of the real-world testing area.

Test Scenario and Safety Properties: We chose a specific situation for the AV to navigate, which involves making a right turn at an intersection where a pedestrian hesitates while crossing a road. The scenario is illustrated in Fig. 5.3. To facilitate execution of this scenario on the test track, we established predetermined starting positions and orientations for both the AV and the pedestrian. We also described the pedestrian's movement as a straight line with three parameters.[2].

- the delay t_{start} after which the pedestrian starts crossing (with a fixed speed of 1 m/s);
- the distance d_{walk} the pedestrian walks before hesitating;
- the amount of time t_{hesitate} the pedestrian hesitates.

[2]Other parametrizations are possible. Our choice here corresponds most directly to what we could implement on the test track

We encoded this scenario as the SCENIC program shown in Fig. 5.3. On lines 7–10 we specify the parameters above to have uniform distributions over appropriate ranges (e.g., $t_{\text{start}} \in (7, 15)$). The functions `DriveTo` and `Hesitate` on lines 3 and 6 specify the dynamic behavior of the AV and the pedestrian, using API calls to Apollo and the LGSVL simulator to command the AV to drive through the intersection and the pedestrian to walk as described above.

Finally, we defined specifications for VERIFAI to monitor during execution of our test cases. The most important safety specification is the following: "the AV never comes close to colliding with the pedestrian." We can formalize this in Metric Temporal Logic (MTL) [94] as $\varphi_{\text{safe}} = \mathbf{G}(\mathit{dist} > 2.5\text{ m})$ where *dist* represents the distance between the AV and the pedestrian (which we can record using the LGSVL Simulator API), and $\mathbf{G}$ is the MTL operator "globally," which asserts that a condition holds at every time point. We chose a threshold of 2.5 m because we measured *dist* as the distance from the center of the AV to the center of the pedestrian[3]: the distance from the center of the AV to its front bumper is 2.1 m and to its side is 0.95 m.

Identifying Safe and Unsafe Test Cases: Having defined the SCENIC program $P_{\mathcal{S}}$ above, we used VERIFAI to perform falsification, sampling parameter values from the distribution $\mathcal{S}$, running the corresponding tests in the LGSVL Simulator, and monitoring for violations of our specification φ_{safe}. We generated 1294 test cases, of which 2% violated φ_{safe}. VERIFAI's error table stored the parameter values for all such cases, along with the quantitative satisfaction value ρ of the specification; for φ_{safe}, this is simply the minimum distance between the AV and the pedestrian over the course of the test, minus 2.5 m. We configured VERIFAI to store the safe runs as well to distinguish robustly-safe runs from near-accident runs. The ρ values help to identify marginal regions that are good candidates for testing. Fig. 5.4 shows ρ as a function of the start delay t_{start} and the walk distance d_{walk}. The darker the points, the smaller the values of ρ, i.e. the closer they are to a collision. We can see that there is no simple relation between parameter val-

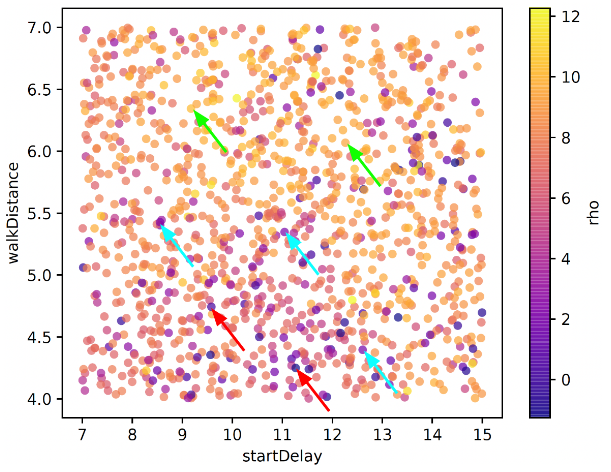


Figure 5.4: Satisfaction value ρ of φ_{safe} (i.e. the minimum distance from the AV to the pedestrian, minus 2.5 m) as a function of t_{start} and d_{walk}. The selected test cases used in the experiment are pointed out with arrows. The arrow colors represent test cases that induce failure (red), marginally safe (cyan), and safe (green) system behaviors.

[3]In future work we plan to improve the simulator interface to measure the distance from the surface of the AV to the surface of the pedestrian.

Test Case	Hesitate Time (s)	Walk Distance (m)	Start Delay (s)	Minimum Distance (m)
F1	2.67	4.50	10.54	2.23
F2	2.93	4.24	11.53	1.91
M1	2.13	4.23	8.50	4.05
M2	1.96	5.02	8.77	4.78
M3	1.03	4.92	9.97	5.85
S1	2.85	6.88	7.64	5.45
S2	2.50	6.33	8.39	5.95

Table 5.1: Track Test Cases Selected from Simulation

ues and collisions that could be determined with a few manually-selected tests: systematic falsification was crucial for test generation.

Test Case Selection: In our experiments, the parameter vector $\vec{\pi}$ was low-dimensional enough for direct visualization (there being only 3 parameters). We observe in Fig. 5.4 that there is one main cluster of unsafe runs, in the bottom-left, and other unsafe runs towards the right for large values of t_{start}; however, the latter were harder to implement due to limitations of track equipment. Using Fig. 5.4 and similar plots for the hesitate time t_{hesitate}, we selected values of $\vec{\pi}$ corresponding to three kinds of tests: failure/unsafe (F), marginally safe (M), and robustly safe/success (S). The success cases were selected from the upper-left quadrant of Fig. 5.4 and have a neighborhood of safe tests. The failure and marginal cases were selected from the bottom-left quadrant. The marginal cases are those that satisfy φ_{safe}, but lie close to other failure cases; hence, implementing these cases in the real world may result in failure due to imprecision in implementing $\vec{\pi}$ on real hardware. We thereby obtained 7 test cases to execute on the track as shown in Table 5.1.

5.3.1 Experimental Setup

Test AV: The test vehicle is a 2018 Lincoln MKZ Hybrid (shown in Fig. 5.5) enhanced with DataSpeed drive-by-wire functionality and several sensors including a Velodyne VLS-128 LiDAR, three Leopard Imaging AR023ZWDR USB cameras, and a Novatel PwrPak7 dual-antenna GPS/IMU with RTK correction for $\sim$2 cm position accuracy. The tests were performed using the open-source Apollo 3.5[4] self-driving software [10] installed on an x86 Industrial PC with an NVIDIA GTX-1080 GPU. Apollo's perception processes data from the LiDAR sensor using GPU-accelerated deep neural networks to identify perceived obstacles.

Pedestrian Dummy and Associated Hardware: To implement the pedestrian at GoMentum, we used a pulley-based 4Active surfboard platform (SB) [1]. The battery powered system drives a motor unit that pulls a drivable platform upon which a "soft target", i.e., an articulated pedestrian dummy [1], is mounted. The dummy is designed to have a sensor signature

[4]This was the most recent Apollo version supported by the Lincoln MKZ vehicle.

similar to real pedestrians. The SB can be programmed for various types of motions, including the "hesitating pedestrian" trajectory used in our scenario.
Triggering Mechanisms: The trigger mechanism of the SB initiates the movement of the pedestrian. For repeatability of scenario testing, it is critical that the same trigger mechanism is implemented both in simulation and in real world. We originally attempted to configure the SB to trigger automatically when a desired distance d_{start} between the AV and the pedestrian is met.

However, the SB manufacturer confirmed that the SB does not support triggering based on distance threshold, and we experimentally confirmed that manual triggering based on an estimate of d_{start} is not accurate. Therefore, we reparametrized our scenario in terms of a threshold *delay* t_{start} measured as the time elapsed from when the AV begins to move to when the pedestrian begins to move. Although the SB hardware does not support automatic triggering based on a time delay either, we were able to implement more accurate triggering by starting a countdown timer when the AV begins to move and manually triggering the SB when the timer expired.

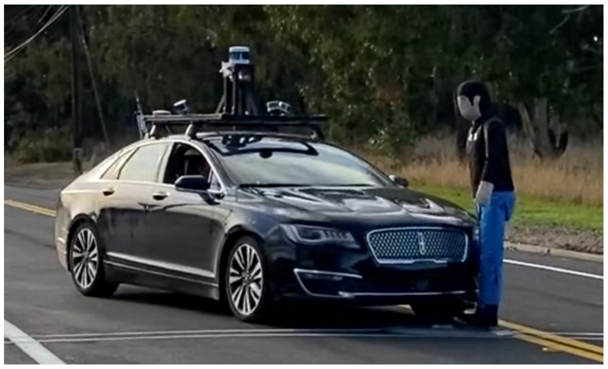


Figure 5.5: The autonomous vehicle and pedestrian dummy used for track testing. The picture shows the AV hitting the pedestrian during testing (test F1 Run 1, https://youtu.be/PehgLCGHF5U), see Section 5.3.1 for details.

Setting up track tests was a tedious and time-consuming effort: it took about 8 people half a day (4 hours) to simply set up the scenario and calibrate the AV and equipment, and then another half a day to go through around 25 test runs, with each run taking 10-15 minutes.

5.3.2 Test Results

We executed the 7 test cases whose parameters are shown in Table 5.1 at GoMentum. We performed several runs of each test scenario, obtaining 23 runs in total; these are summarized in Table 5.2. The highlighted rows in the table are runs which violated φ_{safe} (i.e., have $\rho < 0$), while the white rows satisfied φ_{safe}. The colors of the highlighted rows indicate the degree of unsafe behavior of the AV: red represents a collision (see Fig. 5.5), orange represents what we visually classified as a near-collision, and yellow violates φ_{safe} but is not a near-collision. This coloring scheme brings out distinctions that are not obvious from the Table 5.2 column values. In particular, when φ_{safe} is violated, the pedestrian can be approaching the car from its side or from the front: since the car is much longer than it is wide, a violation of φ_{safe} from a side approach is not always a (near-)collision, resulting in the yellow rows in Table 5.2.

Table 5.2: Test cases selected from simulation, with corresponding runs at GoMentum. Runs violating φ_{safe} are highlighted, classified into collisions (red), near-collisions (orange), or runs which were unsafe but with a larger margin (yellow).

Test Run	Minimum TTC (sec)	Minimum Distance (m)	ρ
F1 Simulation	–	2.23	-0.27
F1 Run1	2.10	2.06	-0.44
F1 Run2	1.27	2.24	-0.26
F1 Run3	2.97	4.02	1.52
F1 Run4	5.05	6.19	3.69
F2 Simulation	–	1.91	-0.59
F2 Run1	0.94	2.44	-0.06
F2 Run2	2.70	3.24	0.74
F2 Run3	1.20	1.58	-0.92
F2 Run4	1.05	2.24	-0.26
M1 Simulation	–	4.05	1.55
M1 Run1	6.07	7.20	4.70
M1 Run2	7.16	7.89	5.39
M2 Simulation	–	4.78	2.28
M2 Run1	3.24	3.40	0.90
M2 Run2	6.16	8.01	5.51
M2 Run3	9.10	14.38	11.88
M2 Run4	6.80	8.05	5.55
M2 Run5	7.69	8.48	5.98
M3 Simulation	–	5.85	3.35
M3 Run1	0.75	1.94	-0.56
M3 Run2	6.00	6.36	3.86
M3 Run3	4.27	5.73	3.23
S1 Simulation	–	5.45	2.95
S1 Run1	1.32	2.79	0.29
S1 Run2	9.72	8.50	6.00
S1 Run3	9.35	7.85	5.35
S2 Simulation	–	5.95	3.45
S2 Run1	3.13	6.36	3.86
S2 Run2	8.66	9.00	6.50

Both in the simulator and at GoMentum, the minimum distance is computed from the *center* of the AV to the pedestrian. Hence, the minimum distance is greater than zero even though a collision occurred in F1 Run1 in Table 5.2. The time-to-collision (TTC) is approximated by dividing the distance between the AV and the pedestrian by the AV's relative speed at every timestamp until either the pedestrian fully crossed the lane or the AV intersected the pedestrian path before it crossed the lane. Videos of all our tests are available at http://bit.ly/GoM_Videos, with a visualization of Apollo's perception and planning at http://bit.ly/DRV_Videos. We will discuss the causes of these failure cases in Sec. 5.3.5.

5.3.3 Analysis

We first consider whether formal simulation was effective at designing track tests that revealed unsafe behaviors of the AV. From Table 5.2, we can see that this was in fact the case: out of 8 runs of the two failure tests that were identified in simulation, 5 violated φ_{safe} in reality, including one actual collision. For example, in test F1 Run1, the AV initially braked, before repeatedly inching forward while the pedestrian hesitated, ultimately colliding with it as shown in Fig. 5.5. More noticeably, in 93.3% of all (marginally) safe runs, AV satisfied φ_{safe} in reality. As expected, a violation case occurred in a marginally safe test, but none in safe tests. While the number of runs is small due to limited time and resources, they clearly demonstrate how simulation can be efficiently used to identify real-world failures.

On the other hand, Table 5.2 also shows that the results of a test on the track can deviate significantly from results in simulation. For example, our first run of test case M3, which was safe in simulation, yielded a very near miss, with a minimum distance of 1.94 m (vs. 5.85 m in simulation). The track test results also have significant variability, with test case M2 for example having minimum distances ranging widely from 3–14 m in different runs. In the next section, we look at these discrepancies in more detail.

5.3.4 The Gap Between Simulation and Reality

There are a variety of possible sources of such discrepancies between simulation and track runs, including:

- mismatch in the initial conditions of the test (e.g. the AV starting at a different location due to GPS error);
- mismatch in the dynamics of the AV or pedestrian (e.g. incompletely-modeled physics in the simulator or imprecision in the mechanism triggering the SB);
- mismatch in the AV's sensory input (e.g.reduced LiDAR point cloud density in simulation, or synthetic image rendering);
- timing differences due to Apollo running on different hardware in the AV and our simulation setup.

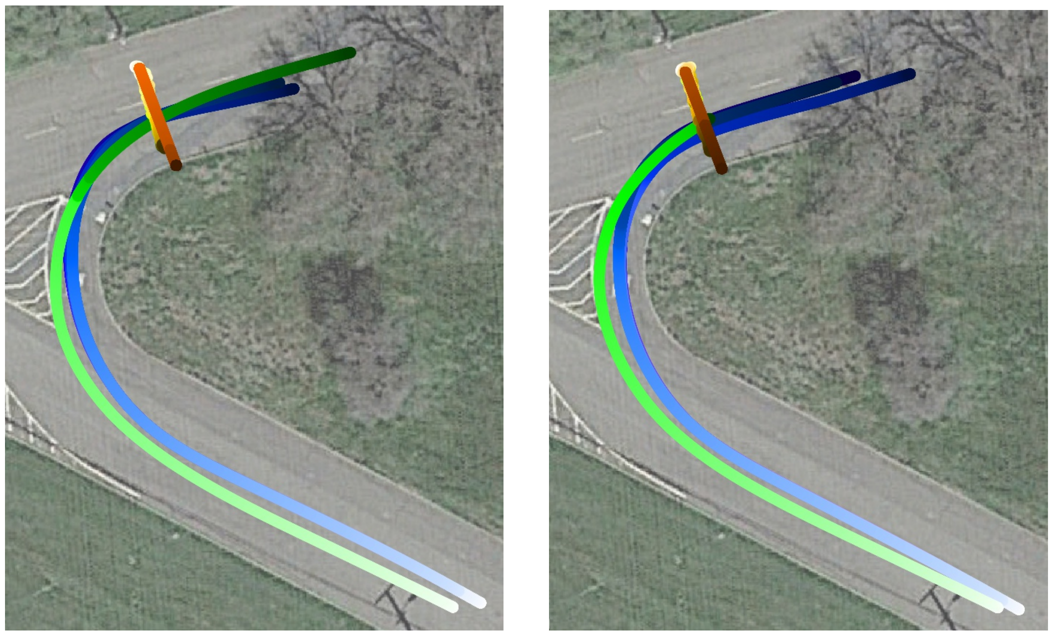


Figure 5.6: Trajectories from a track test and 5 re-simulations respectively for the AV (green/blue) and the pedestrian (orange/yellow). Color darkness indicates time. Scenarios: S1 Run 2 (left), F1 Run 1 (right).

Some of these sources of variation could be eliminated with improvements to the experimental setup which were not possible here given resource and time constraints, e.g., hardware that permits automating triggering based on t_{start} or d_{start} can reduce error in implementing $\vec{\pi}$. However, other potential sources of error are hard to quantify: even small details of rendering, for example, could potentially change the behavior of the perception components in Apollo. To measure the sim-to-real gap resulting from such sources, we used traces recorded from several tests to extract implemented $\vec{\pi}$ on the track. We then ran a new simulation using these $\vec{\pi}$, which would ideally reproduce the same trace as the track test; in fact, we ran 5 identical simulations per test to assess the nondeterminism of the hardware and AV stack.

The resulting trajectories for 2 test cases are shown in Fig. 5.6. The simulated and real trajectories show considerable overlap but also differ: for example, although we intended the simulated AV to start from the same position as the real one, Apollo refused to drive from that position and we had to adjust it slightly. More interestingly, the simulated AV turns more sharply than the real one, possibly due to imprecise modeling in the simulator of the effects of driving slightly uphill. Finally, the 5 simulations are tightly-clustered but not identical, showing that even a single test case can yield a range of different simulated traces.

To quantify these discrepancies, taking into account not only the shape of the trajectories but also their evolution over time, we used the Skorokhod metric, which measures the worst-case deviation of two timed traces and can be used to prove conformance: a bound on how far simulated traces can be from real ones in the Skorokhod metric allows transferring temporal

Track Test	Real-to-Sim		Sim-to-Sim	
Run	Skorokhod	DTW	Skorokhod	DTW
S1 Run2	5.85	1.09	1.11	0.17
S3 Run3	5.05	0.91	2.83	0.24
F1 Run1	10.88	1.39	3.67	0.37
F1 Run3	5.08	0.90	3.29	0.26

Table 5.3: The average distances between track/resimulated AV trajectories. Measures use the L_2 norm with units of meters and seconds.

logic guarantees from simulation to reality [40]. To illustrate the metric, two otherwise-identical trajectories which differ at one point by 1 m or are globally shifted by 1 s would have a Skorokhod distance of 1. We also use the (normalized) Dynamic Time Warping (DTW) distance [65] to give a measure of similarity averaged over the entire trajectory rather than at the worst point. For example, two trajectories differing by 1 m at a single point (out of many) would have a normalized DTW distance of approximately 0, vs. 1 for trajectories differing *everywhere* by 1 m.

Table 5.3 shows these measures for the 4 track tests where we successfully logged accurate GPS trajectories. The "Real-to-Sim" column shows the average distance between the real trajectory and the 5 corresponding simulations, while the "Sim-to-Sim" column shows the average distance among the 5 simulations. The Skorokhod distance between the real and simulated trajectories is large, indicating deviations on the order of 5–11 meters or seconds. This suggests that while our simulation environment was faithful enough to reality to produce qualitatively similar runs, it is not yet accurate enough to enable deriving guarantees on the real system purely through simulation.

Although the simulations are much closer to each other than to the original track test, they still show substantial variation (e.g., the resimulations of F1 Run 1 have large Skorokhod distance). As the LGSVL Simulator is deterministic, this shows that either the asynchronous interface to Apollo or nondeterminism within Apollo itself can produce significantly different behavior on identical test cases. Our methodology could likely be improved by taking this nondeterminism into account: tests with lower variance are more likely to be reproducible on the track, while tests with high variance may indicate undesirable sensitivity of the AV stack.

5.3.5 Why Did the Autonomous Vehicle Fail?

Finally, by replaying our track data in Dreamview, a tool to visualize Apollo's perception, prediction, and planning, we identified several types of failures that led to unsafe behavior, summarized in Table 5.4. The simplest type was a perception failure, where Apollo failed to detect the pedestrian for at least 1 s: this was responsible for the crash in Fig. 5.5. The most common failure was unsafe planning, where Apollo alternated between yielding and

Test Case	Perception Fail.	Prediction Fail.	Planning Fail.
F1 Run1	✓	–	–
F1 Run2	–	–	✓
F2 Run1	–	✓	–
F2 Run3	–	–	✓
F2 Run4	–	–	✓
M3 Run1	–	–	✓

Table 5.4: Hypothesized causes of the observed unsafe behavior.

overtaking the pedestrian. Most interestingly, we also observed a case of prediction failure, where the AV incorrectly predicted that the pedestrian would walk around the AV and that moving forward was, therefore, safe. While our goal is not to find fault with Apollo (recall also that we could only run version 3.5 from January 2019 on our hardware), our results illustrate how our methodology can help to find and debug failure cases.

5.4 Bibliographic Notes

Scenario-based testing of AVs is a well-studied area. One approach is to construct tests from scenarios created from crash data analysis ([125, 25]) and naturalistic driving data (NDD) analysis ([95, 143]), which leverage human driving data to generate test scenarios. Similarly, the PEGASUS project [190] focuses on (i) bench-marking human driving performance using a comprehensive dataset comprising crash reports, NDD, etc., and (ii) characterizing the requirements that AVs should satisfy to ensure the traffic quality is at least unaffected by their presence. Our work differs from these in the use of formal methods for specifying scenarios and safety properties, as well as in automated synthesis of test cases.

Recent work on the test scenario library generation (TSLG) problem ([52, 51]) mathematically describes a scenario, defines a relevant metric, and generates a test scenario library. A critical step in TSLG is to construct a surrogate model of an autonomous vehicle. The authors construct this based on human driving data, which, while useful, may not capture the subtleties in complex ML/AI-based autonomous vehicle stacks. Additionally, the work presents only simulation results, whereas our work reports on both simulation and track testing with a real AV. Abbas et al. [3] present a test harness for testing an AV's perception and control stack in a simulated environment and searching for unsafe scenarios. However, in the absence of a formal scenario description language, representing an operational design domain (ODD) becomes tedious manual labor and challenging as the number of traffic participants scales up.

Researchers have considered the gap between simulation and road/track testing. A methodology for testing AVs in a closed track, as well as in simulation and mixed-reality settings, is explored in [9]. The main aim there is to evaluate the AV's performance across the different settings using standard tests [39], rather than use computational techniques to generate tests based on formally-specified scenarios and outcomes, as we aim to do. A recent SAE EDGE research report [16] dives deeper into unsettled issues in determining appropriate modeling fidelity for automated driving systems. While it raises important questions, it does not address formal methods for evaluation as we do.

Hence, to the best of our knowledge, this work is the first to apply a formal methods-based approach to evaluating the safety of ML-based autonomous vehicles spanning formal specification of scenarios and safety properties, formal simulation-based test generation and selection for track testing, as well as evaluation of the methodology in both simulation and the real world, including systematically measuring the gap between simulation and track testing.

5.5 Chapter Summary

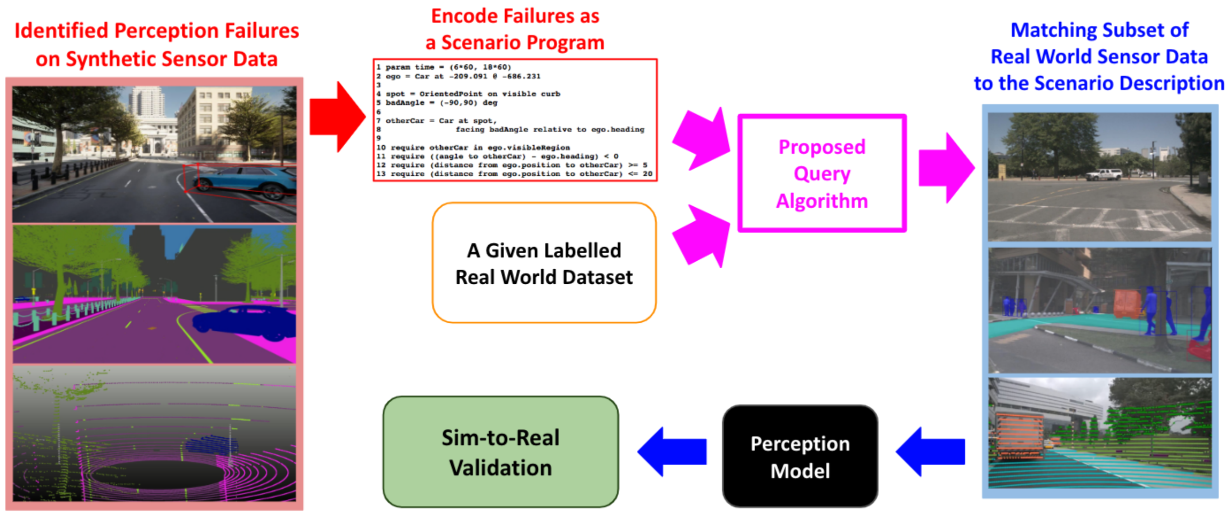
We presented a formal methods approach to scenario-based test generation for autonomous vehicles in simulation, and to the selection and execution of road tests, leveraging the Scenic language and the VerifAI toolkit. We demonstrated that a formal simulation approach can be effective at identifying relevant tests for track testing with a real AV. We also compared time-series data recorded in simulation and on the track, both quantitatively and qualitatively. There are several directions for future work, including evaluating our methodology on more complex higher-dimensional scenarios, performing more detailed automated analysis of failures in perception, planning, or prediction, bridging the sim-to-real gap with further improvements to simulation technology, and developing more sophisticated track test equipment that can better match simulation.

Chapter 6

Querying Sensor Data with Scenario Programs

Although track testing jointly considers different sources of discrepancies between simulation and reality, it is not scalable due to its labor intensive aspect. To enhance the scalability of sim-to-real validation, we restrict our attention to a single dimension of the discrepancies and focus on relaxing the need for physical reconstruction of environments. In particular, *we focus on sim-to-real validation for sensor data discrepancy with application to validating performances of DNN-based perception.* The challenge here is to algorithmically compare the *contents* of environments represented in synthetic versus real sensor data. For example, suppose we identify in simulation that an autopilot fails to perceive another vehicle abruptly cutting into its lane. If we can identify a set of real sensor data with the same contents of such cut-in behaviors, then we can test the perception model on those real data for sim-to-real validation. This data-driven approach does not require physical reconstruction of scenarios as in track testing. Given that massive amounts of real sensor data are collected and labelled [32, 41], the challenge is to use Scenic as an environment formalism to query the labelled dataset to output a subset of real sensor data with contents of interest.

Hence, in this chapter, we develop a scalable sim-to-real validation algorithm for perception models of CPS. The algorithm uses Scenic as a query language to model environments and retrieve sensor data from a provided big, labelled real sensor dataset that matches the scenario description. An envisioned use case of this algorithm is visualized in Fig. 6.1. Developers could encode system failure scenarios identified in simulation as a Scenic program. Using our algorithm, they can query a given labelled real sensor dataset to identify a subset of the real data and test the perception model on the queried subset for validation. If the query returns an empty subset, then we have identified a *rare* scenario not present in our corpus. Hence, our work enables simulation-based testing to guide both real-road testing and data collection by identifying which rare scenarios to focus on. We first explain our assumptions of the dataset and then explain the details of the algorithm.


Identified Perception Failures on Synthetic Sensor Data
Encode Failures as a Scenario Program
A Given Labelled Real World Dataset
Proposed Query Algorithm
Matching Subset of Real World Sensor Data to the Scenario Description
Perception Model
Sim-to-Real Validation

```
param time = (6*60, 18*60)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = (-90,90) deg

otherCar = Car at spot,
                facing badAngle relative to ego.heading

require otherCar in ego.visibleRegion
require ((angle to otherCar) - ego.heading) < 0
require (distance from ego.position to otherCar) >= 5
require (distance from ego.position to otherCar) <= 20
```

Figure 6.1: Using our query algorithm to validate whether a failure of an AV's perception system previously identified in simulation persists in reality.

6.1 Assumptions about the Dataset for Querying

We leverage two technological trends in the AV domain. First, both in academia and industry, there are active community efforts to collect and share large amounts of real sensor data with high quality *labels* [197, 118, 185, 163]. Second, AV companies are *mapping* road infrastructure, gathering detailed geometric information about intersections, lanes, sidewalks, etc. Several such efforts have produced open-source datasets where real sensor data is synchronized with map information, such as nuScenes [22] and Argoverse [30]. Given these trends, we assume we have access to *a large set of labelled real sensor data and the corresponding map information.*

To use our methodology in the context of validating perception behaviors over synthetic and real sensor data, the sensor data must be adequately labelled. Specifically, at minimum the labels must contain information about the position, orientation, and type (e.g. pedestrian, car, bicycle) of each object. Furthermore, the label information needs to be aligned with the map information (i.e. share the same coordinate system and global orientation). Labels may also include more diverse global or object-specific semantic features, allowing more expressive descriptions of scenes. For example, global semantic features could include weather and time of day (which determines the location of the sun and, therefore, of shadows). Object semantic features could include color and physical dimensions.

To query AV scenarios involving road geometry, we require corresponding map information: specifically, polygons for regions such as lanes, roads, and intersections, as well as the traffic flow directions within those regions. This information is needed to interpret SCENIC constructs such as `on road`, seen in our example above. More detailed information may be needed depending on the scenes one may model. For example, if one wishes to model scenes involving cyclists on a bicycle lane in a Scenic program, a polygonal region and traffic flow direction for bicycle lanes in the map must be provided. For more details, see Sec. 2.1.

The above two assumptions are needed for querying labels. However, for our motivating application of validating a failure scenario of a perception model, the real labelled dataset should also contain relevant ground truth labels to evaluate the perception task (e.g. segmentation, detection). For example, if the task is 3D detection, the 3D bounding boxes should be included in the label. Then, once we retrieve a matching subset of real data, the perception model can be evaluated on this subset with the relevant ground truth labels.

6.2 Background: Satisfiability Modulo Theories

The satisfiability problem is the question of whether a propositional formula has a solution, i.e., whether there is an assignment to the Boolean variables that makes the formula evaluate to true. Over the last three decades, satisfiability solvers have made tremendous progress despite the theoretical hardness of the problem, scaling to formulas with billions of variables [117].

To support problems that involve, e.g., arithmetic, SMT solvers ask for a given a first-order formula φ over a set of variables and a background theory[1] whether there is an assignment to the variables that makes the formula evaluate to true [14]. We use a fragment of the theory of quantifier-free nonlinear real arithmetic, with formulas generated by the following grammar:

$$\begin{aligned} e &::= x \mid a \mid e + e \mid e \times e \mid -e \mid \mathit{sin}(e) \\ c &::= e < 0 \mid e \leqslant 0 \mid c \wedge c \mid c \vee c \mid \neg c \end{aligned}$$

Here, x is a real-valued variable, a a real-valued constant, and all operators have their standard meanings. The satisfiability problem for such formulas is undecidable in general (due to the presence of trigonometric functions) [140]. However, solvers such as dReal [64] can either prove unsatisfiability or return a variable assignment which *approximately* satisfies the formula (in a suitable formal sense; see [64] for details). Thus, we do not consider the question of decidability further.

6.3 Problem Statement

Let a *label* l consist of (1) a set of objects O, (2) a set of semantic features F, and (3) a function $s\colon O \times F \to V$ which maps an object's semantic features to concrete values. The domain of values V can include real numbers, integers, categorical values (for examples, see Sec. 6.4.1) and the special value $\perp$, which indicates that the object does not have the

[1]A theory fixes the interpretation of the operators, e.g. $+$ meaning addition and $|x|$ denoting the absolute value of x.

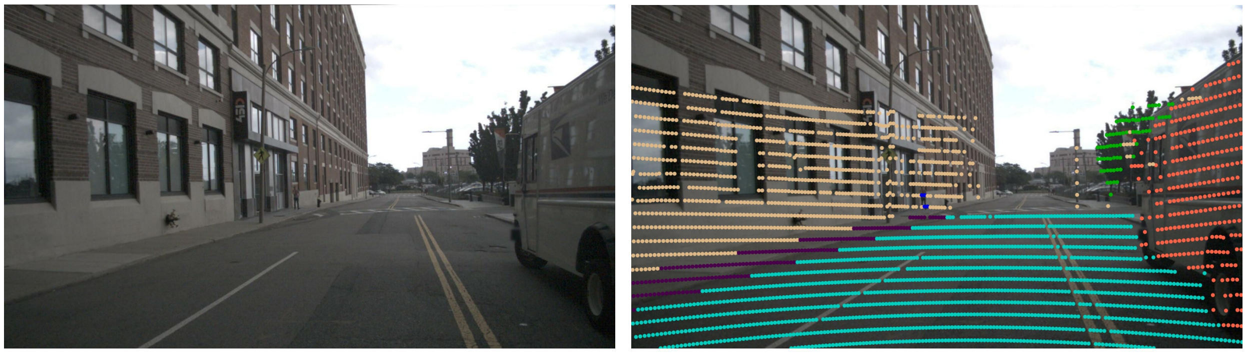


Figure 6.2: A benefit of querying a label is that we can retrieve corresponding sensor data of multiple types. Here, our algorithm finds both an RGB image and a segmented 3D LiDAR point cloud corresponding to a single label.

corresponding semantic feature. Let $[\![P]\!]$ denote the *support* of a Scenic program P, i.e., the set of labels that can be generated by P. Then, the problem to solve could be:

Problem P0: *Given a* Scenic *program P and a label l, is $l \in [\![P]\!]$?*, i.e., is the probability (density) of l under the distribution defined by P greater than 0? However, the above problem statement is too strict for our purposes. In particular, it may answer "no" for a label $l = (O, F, s)$ which is not in $[\![P]\!]$ even if an essentially equivalent label $l' = (O', F', s')$ *is* in $[\![P]\!]$. This can happen in two ways: (1) The semantic feature spaces of l and l' may differ, i.e., $F \neq F'$. For instance, a car's color and model might be included in F but not in F'. In such cases, we will only consider features in $F \cap F'$ as being relevant to our query. (2) More importantly, l may contain multiple objects, and the correspondence between O and O' is unknown; in fact, we may want to consider l as matching P even if it contains additional objects not having any counterpart in P (for example, we may want a program for "two perpendicular cars" to match whenever two such cars exist, even if there is a third car in the vicinity).

Therefore, we generalize **P0** as follows.

Problem P1: Let a label l *match* a program P if there exists $l' = (O', F', s') \in [\![P]\!]$ and an injective[2] function $C\colon O' \to O$ such that $\forall o' \in O'$, $\forall f \in F \cap F'$, $s'(o', f) = s(C(o'), f)$. Here the function C, which we call the *object correspondence*, maps each object in P to a distinct object in l. Then, *Given program P and label l, determine whether l matches P.* This definition supports matching noisy labels: simply modify the Scenic program to add bounded perturbations or noise models to any semantic features that might be noisy.

Our problem formulation can be applied to validate the behaviors of models for different perception tasks using various sensor types (e.g. LiDAR segmentation, RGB detection) across synthetic and real data. Because we query the labels, our problem formulation broadly applies to querying various labelled sensor types such as 2D RGB, radar, 3D LiDAR point

[2]If we want to require an *exact* match in the sense that l does not contain any objects beyond those defined in P, we can also require C to be surjective. Our algorithm extends to this case with only trivial changes.

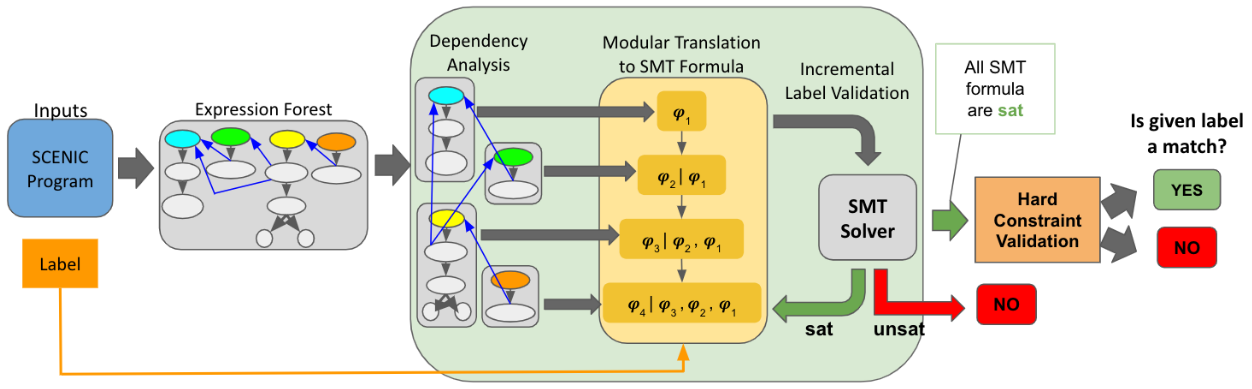

Inputs
SCENIC Program
Label
Expression Forest
Dependency Analysis
Modular Translation to SMT Formula
φ_1
$\varphi_2 | \varphi_1$
$\varphi_3 | \varphi_2, \varphi_1$
$\varphi_4 | \varphi_3, \varphi_2, \varphi_1$
Incremental Label Validation
SMT Solver
sat
unsat
NO
All SMT formula are sat
Hard Constraint Validation
Is given label a match?
YES
NO

Figure 6.3: An overview of our algorithm to determine if a label matches a Scenic program.

clouds, etc. For example, it is possible that for the same label, data from multiple sensors exists. Fig. 6.2 shows 2D RGB image and 3D LiDAR point cloud data that corresponds to the same label. As we query labels rather than the raw sensor data, we can query both sensor types.

6.4 Methodology

Prior to explaining our scenario query algorithm, we provide an overview first to facilitate understanding.

6.4.1 Overview

Suppose we wish to query a label against the simple Scenic program in Fig. 6.4. The program describes, for some fixed map with information about roads and intersections, a scenario where there is a car ahead of the ego vehicle but not in an intersection. A label consists of semantic features such as time of day, weather conditions, and positions and orientations of vehicles. For this particular example, assume that our (simplified) label l consists of features $l_{e_x}, l_{e_y}, l_{e_h}, l_{e_{cl}}$ denoting the xy-coordinates, heading, and object class (e.g. car or pedestrian) of the ego car respectively, and, likewise, features $l_{c_x}, l_{c_y}, l_{c_h}, l_{c_{cl}}$ for the other car.

When we query a label against this program, we want to answer the question: *is the situation specified by the label an instance of the scenario in Fig. 6.4?* More precisely, we ask whether the label can be obtained by instantiating the random variables in the scenario (i.e. for some choice of the weather, time, ego's position "on road", and a distance between 4–10 meters).

Our approach to this problem is summarized in Fig. 6.3. Before we go into details, let us clarify that in a nutshell, the approach is to translate the Scenic program and the label into constraints represented as a Satisfiability Modulo Theory (SMT) formula (see Sec. 6.2 for

```
param weather = Uniform('sunny','rainy')
param time = Range(10, 12) # pm

ego = Car on road, facing roadDirection
otherCar = Car ahead of ego by Range(4,10) #meters
require not (otherCar in intersection)
```

Figure 6.4: A Scenic program describing a car ahead of the ego car by 4–10 meters and which is not in an intersection.

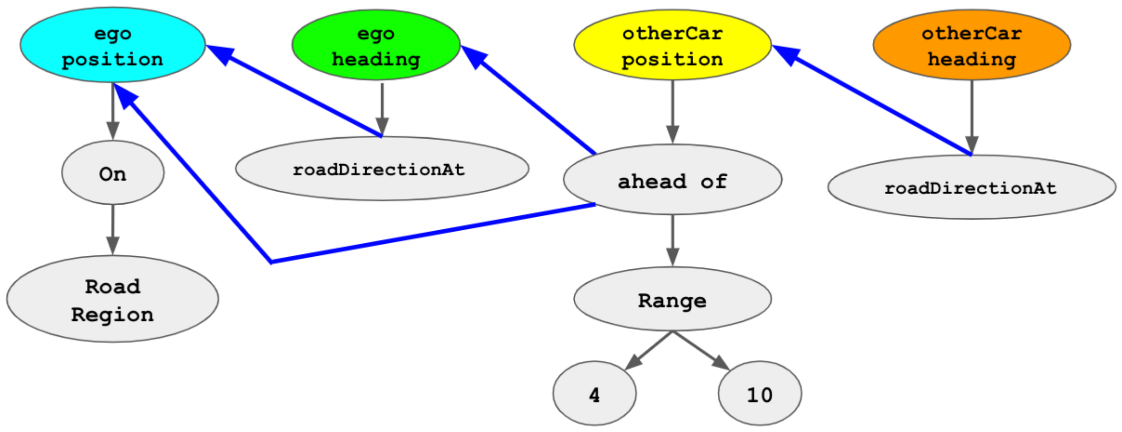


Figure 6.5: A partial expression forest for the Scenic program in Fig. 6.4. The 4 top nodes represent the semantic features, and the blue arrows show dependencies among them.

an overview of SMT). The resulting formula will be satisfiable if and only if the given label matches the program. While the problem can be captured by a single (monolithic) SMT formula, the size of this formula increases linearly with the size of the scenario, and for even relatively simple scenarios can exceed the capabilities of state-of-the-art SMT solvers. To alleviate the scalability problem, we take advantage of the structure of the Scenic program to decompose it into several SMT formulas, determining incrementally whether parts of the label match the program. Below, we discuss the key stages of Fig. 6.3 in more detail.

Expression forest To enable the decomposition process, we operate on the internal representation of a Scenic program. The Scenic compiler converts a program into an *expression forest*: a simplified forest for the program in Fig. 6.4 is shown in Fig. 6.5. The expression forest is made up of a set of expression trees, each of which essentially corresponds to the syntax used to define the distribution of one of the semantic features. For example, the leftmost tree in Fig. 6.5 shows that the ego's position is defined to be a uniformly random point in the `road` region of the map.

Dependency analysis Expression trees can have dependency relations with each other, as shown by the bold blue arrows in Fig. 6.5. Such dependencies naturally occur in scenario modeling. For example, to compute the `otherCar`'s position (which is `ahead of` the ego car), we need to first know the ego's position and heading. Our dependency analysis uses the Scenic expression forest to sort the list of semantic features in dependency order: in Fig. 6.5, the ego's position is first since it depends on no other features, while the `otherCar`'s heading is last.

Modular translation Next, we *modularly* translate each expression tree and *incrementally* check its consistency with the label in dependency order. In our example, we start with the ego's position, symbolically representing its coordinates by variables e_x and e_y. We encode its definition given by the leftmost expression tree in Fig. 6.5 with the constraint $\texttt{On}(\texttt{RoadRegion}, e_x, e_y)$, where `On` is a predicate requiring that the point (e_x, e_y) is contained in the polygon that defines the `RoadRegion`. We can then check whether the value of this feature given in the label is in fact possible in the Scenic program by checking the satisfiability of the SMT formula

$$\varphi_1 = \texttt{On}(\texttt{RoadRegion}, e_x, e_y) \land (e_x = l_{e_x}) \land (e_y = l_{e_y})$$

where as above l_{e_x} and l_{e_y} are the label's concrete values for e_x and e_y, and $\land$ represents a logical AND.

Incremental validation If the formula is unsatisfiable, the observed label cannot be generated by the Scenic program, and so does not match the scenario. If the formula *is* satisfiable, then this first semantic feature is consistent with the scenario and we can move on. To avoid reasoning about this feature again, we *condition* the ego position expression tree by replacing it with the actual position value from the label. This conditioning will apply when we proceed to check the consistency of the next feature in our order, namely the ego's heading. Translating the second expression tree to SMT yields the formula

$$e_x{=}l_{e_x} \land e_y{=}l_{e_y} \land (e_h{=}\texttt{roadDir}(e_x, e_y)) \land (e_h{=}l_{e_h})$$

which we call $\varphi_2|\varphi_1$ to indicate the conditioning. Note that the ego position's expression tree, involving the `On` predicate, is no longer present, simplifying the formula: this is the essence of our incremental approach. We then solve the new SMT formula, and repeat this modular feature translation and incremental validation process until either all the features in the label are validated, or a feature is invalidated. If all the features are valid, then as a final step we condition all the features as in the label and check whether all the explicit constraints in the Scenic program (e.g. the `require` statement in Fig. 6.4) are satisfied. If so, the label matches the scenario; otherwise, it does not.

6.4.2 Our Scenario Query Algorithm

Given a label and a Scenic program, the key idea behind our approach is to translate the program to an SMT formula that is satisfied if and only if the label matches the program.

Monolithic Approach

The basic set-up of the SMT formula defines variables for all objects and semantic features in the intersection of program and the label. There are three aspects to the SMT formula; the first aspect of the formula maps objects in the label with objects in the program (which we referred to as *correspondence* in Sec. 6.3), the second aspect describes the constraints over all the semantic features in the program as dictated by the semantics of Scenic, and the third aspect asserts that the semantic feature values observed in the label satisfies those corresponding constraints in the program.

The Scenic compiler represents a compiled Scenic program as an *expression forest*. Each semantic feature is the root node of an *expression tree* appearing in the forest, which captures the semantics of how the feature's value is derived from the values of other features and random parameters. To encode the semantics of the Scenic program, we walk the expression forest, generating SMT equivalents of each of the nodes. For example, the `On` node in Fig. 6.5 is encoded by a set of constraints enforcing that the variable representing the ego's position must lie in the `road` region[3]. All Scenic constructs[4] can be encoded as real arithmetic constraints fairly easily, following the Scenic semantics outlined in [60]. The details of our SMT encoding and the fragment of Scenic we support are explained in Appendix C.1. This method yields a sizeable SMT formula, which is difficult to solve beyond toy examples. Therefore, we consider a modular approach outlined below.

Modular and Incremental Approach

Our approach is formalized as Algorithm 1. The input to the algorithm is a Scenic program P and a label l, and the output is whether l matches P. The algorithm has three main steps: (1) dependency analysis of objects and their features, (2) incremental translation and validation of the program to a series of SMT formulas, and (3) validation of hard constraints. We now discuss each step in detail.

Object/Feature Dependency Analysis

A key feature of our approach is to exploit dependency structure in the Scenic program, specifically its compiled expression forest, to split the monolithic SMT query into smaller parts. We define two types of dependencies in the expression forest: *dependent* and *jointly*

[3]This is done by triangulating the region, and using a disjunction of linear inequalities to assert that the position lies in one of the triangles. See Appendix C.1 for more details.

[4]Our implementation handles a large fragment of Scenic, supporting its built-in operators and functions. However, it does not support the inclusion of arbitrary Python code, which Scenic allows in some contexts.

Algorithm 1 Determining if SCENIC program P matches label l

```
1:  $EF \leftarrow Compile(P)$ // get expression forest
2:  SortedFeatures $\leftarrow AnalyzeDependencies(EF, l)$
3:  $badOCs \leftarrow \varnothing$ // partial correspondences that don't work
4:  for all possible object correspondences $C$ do
5:      if $C$ extends a correspondence in $badOCs$ then
6:          continue; // skip this correspondence
7:      $failed \leftarrow false$
8:      for $nextFeatures \in$ SortedFeatures do
9:          $\varphi \leftarrow TranslateSMT(nextFeatures, l, C, EF)$
10:         if $Satisfiable(\varphi)$ then
11:             $Condition(nextFeatures, l, C, EF)$
12:         else
13:             $failed \leftarrow true$
14:             $Uncondition(EF)$ // reset forest
15:             add the used part of $C$ to $badOCs$
16:             break
17:     if (not $failed$) and $SatisfiesHardConstraint(l)$ then
18:         return $Yes$
19: return $No$
```

dependent. If an expression tree of a feature, X, has a reference to another feature, Y, then X is *dependent* on Y. Such dependencies are *acyclic* because a feature that is specified first in SCENIC cannot reference an object defined afterwards. If two or more feature expression trees share internal nodes which are not features, then those features are *jointly dependent*. These shared node(s) are intermediate (i.e., unobserved) variables which are not part of the scene. Therefore, to check whether there exists a feasible value for the intermediate variables, jointly dependent features must be considered in the same SMT query. For example, Fig. 6.6 shows a SCENIC program describing a distribution of scenes with two cars positioned in parallel, adjacent to a spot uniformly randomly selected from a curb region. In this case, the ego and side car's position features are *jointly* dependent on the intermediate variable, `spot`. Note that `spot` is an internal variable of the SCENIC program and would not appear in a label. Hence, we need to encode both cars into the SMT query to check if there exists a value of `spot` which satisfies the constraints given their labelled positions. Note that for the following dependency analysis among semantic features, as we stated in Sec. 6.3, we are only analyzing semantic features that exist both in the SCENIC program and the label. Hence, this dependency analysis takes as input both the expression forest of the given SCENIC program and the label (line 1 of Alg. 1).

Once we identify these dependency relations across all object features, we sort the features in dependency order, giving jointly dependent features the same rank in the order (line 2 in Alg. 1). For example, analyzing the expression forest in Fig. 6.5, with no jointly dependent

```
spot = OrientedPoint on curb
ego = Car at (spot offset by (Range(2,4), Range(5,10)))
sideCar = Car left of spot by Range(1,3)
```

Figure 6.6: A Scenic program with an intermediate variable, `spot`, shared between position features of `ego` and `sideCar` objects.

features, yields the order [{`ego` position}, {`ego` heading}, {`otherCar` position}, {`otherCar` heading}]. For the Scenic program in Fig. 6.6 with jointly dependent features, our analysis outputs [{`ego` position, `sideCar` position}, {`ego` heading}, {`sideCar` heading}].

Modular and Incremental SMT Translation

Given the sorted feature dependency list obtained as above, we translate only one feature expression tree (or multiple trees, if jointly dependent) at a time, in the order of dependency (lines 8-9 in Alg. 1). Checking the resulting SMT query, if the formula is unsatisfiable then it is impossible for the current features to take on their observed values and so the label does not match the program (lines 12-13). If instead the formula is satisfiable, then the observed values are feasible given the semantics of the program, and we need not consider the current features further: we *condition* the expression forest on their observed values, substituting the values in for their expression trees (lines 10-11). Then we move on to the next feature(s) in the dependency order and repeat. If a previously-checked feature is referenced by a later expression tree, we do not need to encode it again, since it now has a constant value. This modular approach can significantly simplify the generated SMT queries: for example in Fig. 6.5, instead of one query encoding the entire forest, we have one query for each of the 4 top nodes encoding only the nodes directly below it in the order of dependency (refer to Sec. 6.4.1 for more detail).

We remark that our modular translation requires fixing the correspondence of label and program objects *a priori*. As seen in the outermost loop of Alg. 1 (line 4), we currently brute-force enumerate all possible combinations, with one refinement: if a partial correspondence is already enough to make the SMT query unsatisfiable, we can exclude all further correspondences extending it. In particular, when a feature of object O fails the SMT check under correspondence C, the part of C consisting of all objects up to and including O (in dependency order) will also yield unsatisfiability. So we maintain a set of partial correspondences known to fail (line 15) and skip any correspondence which extends one of them (line 6). Our experiments show that this approach scales to a reasonable number of objects.

A further note is that encoding polygonal regions of the map (e.g. lanes, roads, intersections) can yield large formulas. For example, nuScenes, the dataset used for our experiments, provides a map of the city of Boston. Encoding the entire road network of Boston would be impractical; however, since we are only interested in the scene around the ego vehicle, which determines the reference viewpoint of sensors (e.g. camera, LiDAR), it suffices to encode only a neighborhood of the ego. We can extract bounds on the visible distance from the Scenic program and only encode the region of the map within that radius.

Hard Constraint Validation

Finally, if the Scenic program contains any `require` statements encoding hard constraints (see Sec. 2.1), we need to check that these are satisfied by the label. After all features have been validated and conditioned on their observed values, we simply check that the `require` constraints all evaluate to true. Note that if any `require` constraints were jointly dependent (see Sec. 6.4.2) with any feature, for soundness we would have to encode the constraints into the SMT query for that feature. Since the Scenic compiler currently does not generate expression trees for requirements, we instead assume that the program does not have any such joint dependencies (restricting our Scenic fragment; see Appendix C.1). This assumption holds for the vast majority of the scenarios in the Scenic distribution [60]. Our query algorithm is sound as stated in Theorem 1 (see Appendix C.2 for the proof).

Theorem 1. *Given a label and a* Scenic *program, the* Scenic *query algorithm outputs* `Yes` *if and only if the label matches the program as defined in Sec. 6.3; otherwise, it outputs* `No` *(assuming the underlying SMT solver correctly answers all queries).*

6.5 Experiment

Recall that we motivated the query problem with the application of validating failure scenarios. Once our algorithm queries a labelled, real dataset with a scenario encoded in Scenic and retrieves a matching subset, then validating a perception model's behavior on that subset is straightforward (assuming the dataset also contains the relevant ground truth labels for the perception task: see Sec. 6.1). In fact, the most time-consuming aspect in this validation process is executing the query. Thus, to evaluate how useful our algorithm is, we ask the following questions:

1. Given a Scenic program and a real labelled dataset, does the algorithm efficiently find the matching data points?
2. Does the output of the algorithm correspond with the intuitive notion of scenario matching?
3. How does the algorithm scale with scenario complexity, in terms of number of agents and program structure?

The second question is important to address since it directly relates to the *interpretability* of the scenario validation process, which is crucial for debugging. Therefore, we need to check whether our algorithm operates in a manner intuitive to humans.

To answer these questions, we conducted two different experiments. First, our efficacy experiment answers the first two questions. In a nutshell, it demonstrates that the formal querying problem we define and solve corresponds well to our intuition of what it means to match an image against a high-level scenario and is efficient in comparison to manual querying. Our second experiment demonstrates that our approach remains feasible even on fairly large scenarios. In both experiments, we used the dReal SMT solver [64] with its

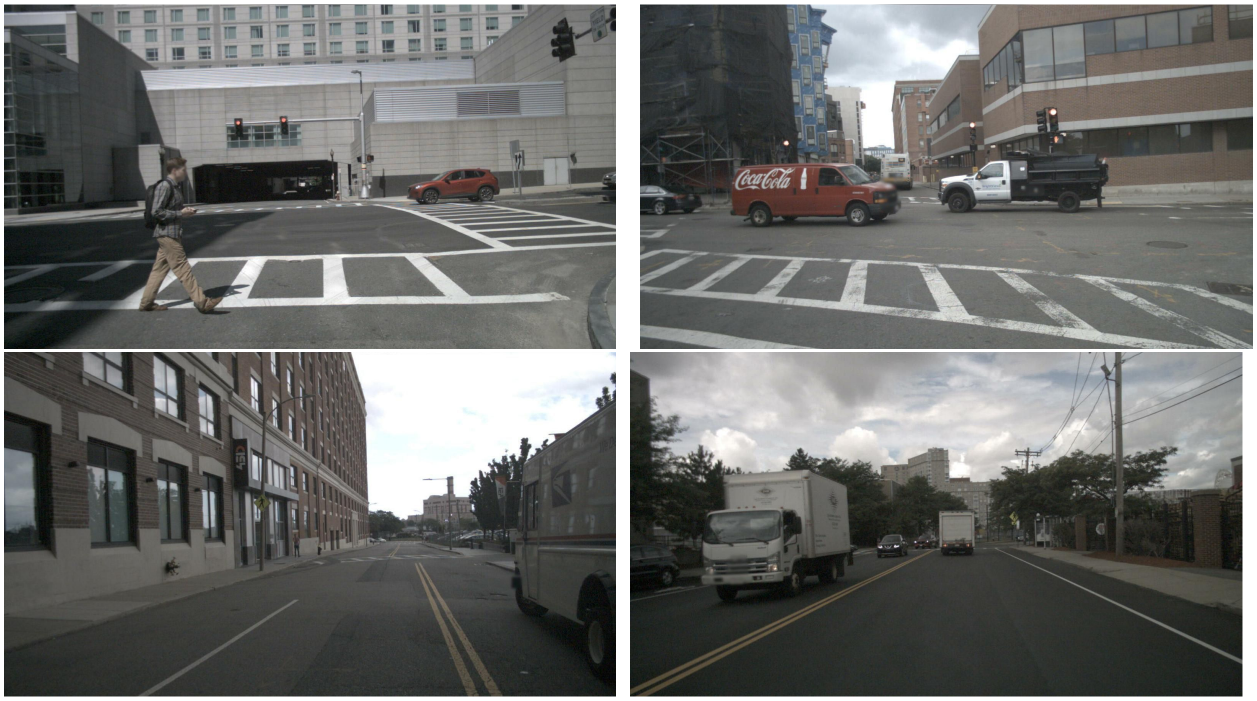


Figure 6.7: Matching images for Scenarios 1 through 4 (left to right), queried using our algorithm.

default parameter settings. For the first experiment, we set the ego visible distance to 50 meters; for the second, we set it to 200 meters to accommodate the larger number of agents.

6.5.1 Efficacy Experiment

Setup There is no baseline or benchmark to which we can compare our algorithm since there are no existing algorithms for the problem of querying with a formal scenario description, or open-source autonomous driving image datasets that provide detailed formal scenario descriptions with which to test our algorithm. Hence, we asked 3 human participants to manually query a set of images with 5 different scenarios and then compared their results with the outputs of our algorithm. We asked each participant to select 5 subsets of images matching more detailed versions of the natural language descriptions of our test scenarios below. To acquire the most accurate queried subsets, we kept only the images which all 3 humans agreed matched for each scenario. We then compared these subsets with those returned by our algorithm.

Scenarios We used five scenarios, involving 2–4 agents and a variety of realistic traffic situations. Here we provide natural language descriptions; the Scenic encodings are shown in Figs. C.5–C.9 in Appendix C. Several example matching images are shown in Fig. 6.7.

1. A pedestrian in an intersection facing nearly perpendicularly or towards the ego.

Scenario #	1	2	3	4	5
Matching images (humans)	42	5	0	2	0
Matching images (our algorithm)	58	7	2	2	0

Table 6.1: For several scenarios, the number of images identified by 3 human subjects (unanimously) and our algorithm.

2. Two vehicles in an intersection, travelling perpendicular to the ego.
3. A rare, hazardous situation, where the ego vehicle is driving against traffic and another vehicle is visible within 10 meters.
4. Four vehicles in a typical situation on a two-lane road, with two vehicles going in each direction.
5. A cut-in scenario where a car in the adjacent lane to the right cuts in front of the ego.

Data We use a selection of RGB images from nuScenes [22]. nuScenes provides the map of Boston where the images were collected; our scenarios used map information about intersection and lane regions as well as traffic flow directions. nuScenes labels contain three semantic features per object: its position, heading, and class. The object classes include vehicles, pedestrians, and static objects such as traffic cones. nuScenes also defines a special object class, "ego vehicle", indicating the reference viewpoint, meaning the camera for image collection is mounted on it. As we will describe below, our experiment required humans to identify matches between programs and labelled images; to avoid objects being missed by the humans due to visual occlusion, we filtered out images containing more than 4 objects. After filtering, we randomly selected 700 images from the subset, which we believe is a reasonably large dataset for human participants to manually query.

Note that the semantic features in the label and the program determines the matching real data. In our experiment, we limited our query to use only semantic features included in the nuScenes labels, namely position, heading, and type for each object.

Results Our results, summarized in Table 6.1, show that our algorithm corresponds to the intuitive notion of scenario matching. For all 5 scenarios, our algorithm correctly returned all images identified by the human participants, and, in some cases, found additional matching images that they missed. For Scenarios 2 and 3, our algorithm found 4 additional images that our participants missed by mistake. An example missed image for Scenario 3 is shown in Fig. 6.7. For Scenario 1, our algorithm identified 16 more images than the participants; however, upon visual inspection, 8 of these did not match the scenario description and the remaining were missed by our participants. Investigating further, we found that these errors were caused by inaccurate labels in nuScenes, e.g. a pedestrian's position being in an intersection according to the label but being on a sidewalk near an intersection in the image. Our algorithm correctly identified such labels as matching Scenario 1, even though the sensor data disagreed. This illustrates a limitation of our approach in that it hinges on

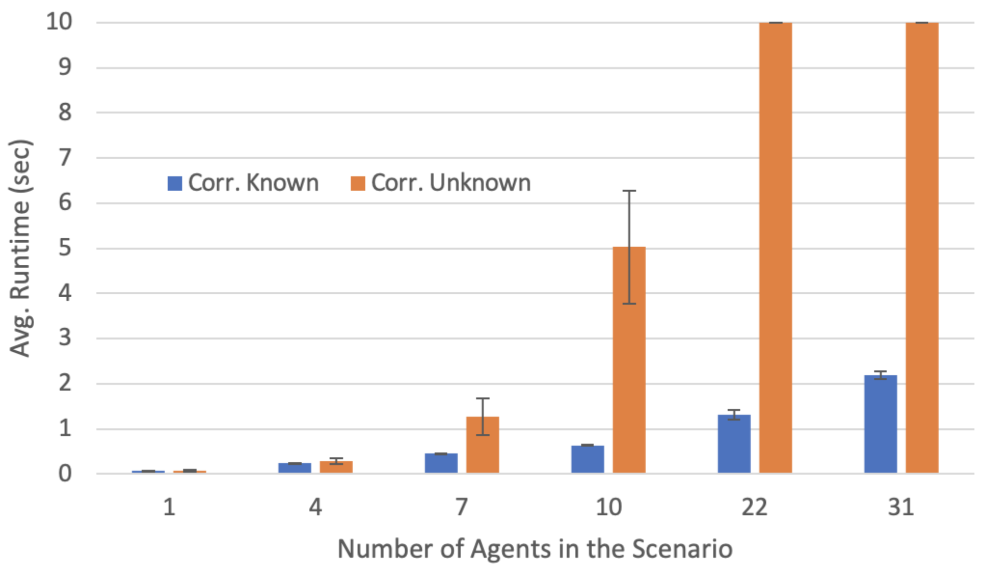


Figure 6.8: Runtime results for scaling number of agents in the SCENIC program shown in Fig. 6.10. Two runtimes represent cases when the object correspondence between the program and the label is known versus unknown.

the accuracy of the provided labels. Finally, for Scenario 5, both the human participants and the algorithm agreed that there are no matching scenes in the given real dataset. This shows the strength of our algorithm in identifying "rare" scenarios with respect to a given dataset. If an AV perception model fails on such rare scenarios in simulation, this finding can guide the real-world data collection process to *systematically* gather more examples of such scenarios.

The algorithm's runtime over all 700 labels ranged from 7 minutes (Scenario 1) to 40 minutes (Scenario 5) depending on the complexity of scenarios. The human participants took on average about 1 hour to complete their manual queries on the five provided scenarios. This result demonstrates that our algorithm can replace the arduous task of manually querying matching real sensor data for scenario validation with higher accuracy, provided that the labels are accurate.

6.5.2 Scalability Experiment

To test the scalability of our algorithm, we used two additional syntactically-rich SCENIC programs, shown in Fig. 6.10 and 6.11. We increased the scenario complexity by scaling the number of agents while maintaining the same program structure. For evaluation, we generated 10 labels from each SCENIC program for a range of numbers of agents (from 1 to 31).

Figures 6.8 and 6.9 show the average runtimes for querying these 10 labels with our algorithm for each number of agents (with a 10-second timeout). In each plot, we give

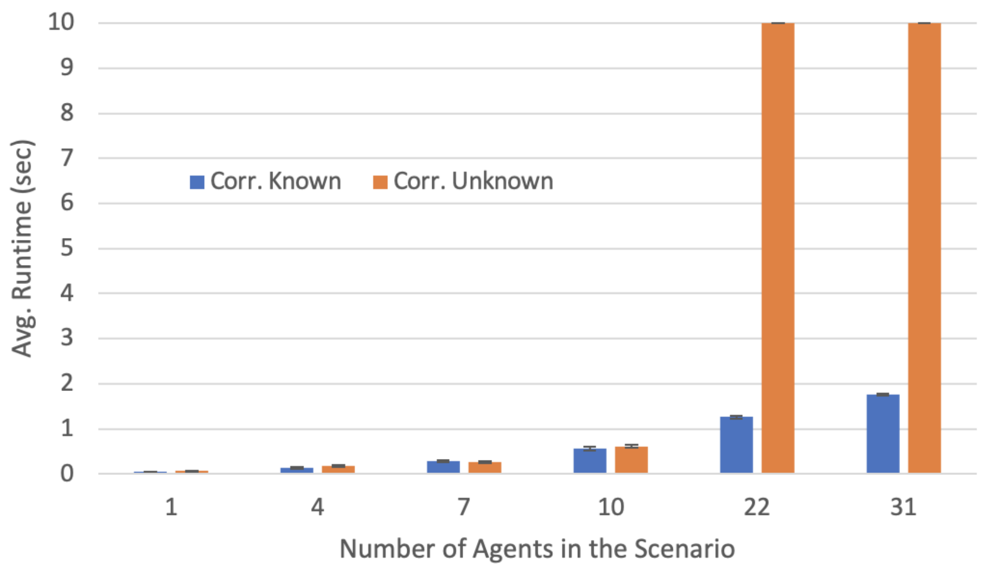


Figure 6.9: Runtime results for scaling number of agents in the SCENIC program shown in Fig. 6.11. Two runtimes represent cases when the object correspondence between the program and the label is known versus unknown.

separate runtimes for the cases where the correspondence between the objects in the SCENIC program and the label is known and unknown respectively. For the unknown case, we randomly shuffled the ordering of the objects in the labels.

We observe a consistent behavior from these two plots. When the correspondence is unknown, the runtime increases exponentially in the number of agents due to the combinatorial object correspondence matching process in our algorithm (refer to Sec. 6.4.2). On the other hand, when the object correspondence is known, we consistently observed that the runtime of the query algorithm increases approximately linearly in the number of agents, and scales to a sizeable number.

6.6 Bibliographic Notes

The most related study to ours is [61], where failure scenarios of an autopilot in simulation were validated by *physically* reconstructing them at a track testing facility. However, this manual validation approach is labor-intensive and not scalable. Our approach aims to automate such validation in a data-driven manner.

Domain adaptation [184] aims to reduce the sim-to-real gap in the context of *training*: the objective is primarily to obtain good performance of a perception model for a particular task (e.g. segmentation, detection, localization) on *real* sensor data despite only training on simulated sensor data [85, 131, 200, 124]. Generative adversarial networks (GANs) [37] have been a key technique employed to *adapt*, or convert, synthetic data to more realistic data of various types such as RGB images, 3D LiDAR point cloud, etc. These data are used

```
def placeObjs(car, numCars):
    for i in range(numCars):
        car = Car ahead of car by Range(4, 5)
        leftCar = Car left of car by Normal(2,0.1),
            facing roadDirection
        rightCar = Bicycle right of car by Normal(3,0.1),
            facing Range(0,10) deg relative to ego.heading
    return leftCar, rightCar

spawn_point = 207.26 @ 8.72
ego = Car at spawn_point,
        with visibleDistance 200

leftCar, rightCar = placeObjs(ego, 2)
require (distance to leftCar) < 200
require (distance to rightCar) < 200
```

Figure 6.10: A Scenic program used for our scalability experiment, modeling bumper-to-bumper traffic. The number of vehicles in the scenario is scaled by increasing `numCars`.

```
def placeObjs(numPeds):
    for i in range(numPeds):
        Pedestrian offset by Range(-5,5) @ Range(0,200),
            facing Range(-120, 120) deg relative to ego.heading

spawn_point = 207.26 @ 8.72
ego = Car at spawn_point,
        with visibleDistance 200

placeObjs(3)
```

Figure 6.11: A Scenic program used for our scalability experiment, modeling a parade scenario where pedestrians are walking on a street. The number of pedestrians in the scenario is scaled by increasing `numPeds`.

for training. On the contrary, our work aims to reduce the gap in the context of *testing*, investigating for a *pre-trained* model whether it behaves differently in the same scenario across two different domains: simulated and real data.

Visual question answering (VQA) [70, 196] considers answering questions about static images phrased in natural language. The VQA area combines approaches common in captioning with a large natural language processing component: part of the challenge is to understand the question. Much like VQA, we decide whether an image matches a given query. However, our queries are expressed using a formal probabilistic programming language and we query the *label*, not the sensor data. This allows us to formulate a well-defined querying problem and develop an algorithm which is guaranteed to be sound: the returned subset of images are exactly those which match the scenario program, if the labels are accurate.

Our approach can be seen as a specialised form of inference in probabilistic programming languages (PPLs). SCENIC allows making probabilistic assertions of propositional statements, e.g., 'the car is within the visible region'. Such declarative *hard constraints* make scenario modeling much easier and more intuitive. Some PPLs, e.g. ANGLUIN, actively prevent specifying hard constraints to prevent programmers from 'accidentally' posing NP-hard questions [192]. PPLs such as PYRO [17] and EDWARD [172] use Bayesian inference schemes that require tracking derivatives [79, 189, 96, 97]. Some PPLs allow hard constraints and non-continuity, but either have (significant) restrictions or limited efficiency [129, 72, 80], putting trigonometry and continuous domains out of reach. Moreover, the typical inference task is to compute posterior distributions relative to a prior, whereas we are primarily interested in filtering using a Boolean membership query. Finally, we note that while sampling-based approaches may do well in answering membership queries positively, they are not well suited for providing negative answers.

6.7 Chapter Summary

We developed an algorithm to query a labeled dataset using a scenario program encoded in the SCENIC language. This algorithm can be used to shrink the gap between simulation-based and real-world testing by identifying counterparts of simulated scenarios in real data, which can then be used to validate the fidelity of the simulations. More broadly, our algorithm enables a principled way to explore and understand the range of scenarios present in a dataset by expressing scenarios of interest in a formal language. In future work, we plan to explore using program analysis and other techniques to alleviate the combinatorial explosion when the object correspondence is unknown, and to generalize our algorithm to support *dynamic* scenarios, as well as *probabilistic* queries that take the likelihood of labels into account.

Part III

Failure-Informed Targeted Training

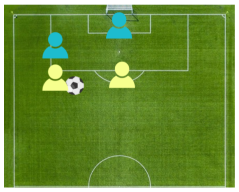
Chapter 7

Programmatic Training for Reinforcement Learning

In Part I and II, we identify failures of autonomous systems or their components in simulation, and then validate them on real sensor data. Now, informed by these failures, how can we actually fix the systems to be robust to them? Previously, there have been work proposed to augment data using domain-specific PPLs as environment modeling formalism to further train and fine-tune DNN-based components, particularly perception [45, 57]. However, the methodology to utilize PPL to efficiently train reinforcement learning (RL) algorithms in online or offline manner has not been investigated. Meanwhile, RL algorithms are widely used in autonomous systems, especially for planning and control of robotics, in increasingly more complex environments. Yet, a PPL has not been used to model and generate stochastic and reactive multi-agent scenarios in RL. In this chapter, we devise methodologies to train RL algorithms in either online or offline manner using a PPL [13].

7.1 Scenario Specification Language for RL

The objective of this chapter is to introduce the benefits of the use of a scenario specification language for modeling and generating scenarios, specifically for real-time strategy (RTS) environments for RL. These RTS environments are characterized by unique characteristics that require special support for modeling. The environments involve intelligent entities/non-RL agents co-operating and competing with the RL agents with large state and action spaces over a long horizon. This opens up extremely diverse strategies consisting of numerous interactive behaviors. Yet, most of the existing simulators rely on randomly generating the environments based on predefined settings/layouts and offer limited flexibility and control to the researchers over the environment dynamics to generate diverse realistic scenarios. As a result, RL research face at least two fundamental challenges: (i) the lack of diverse and realistic training data often leads to lack of generalization [35, 34, 106, 107], and (ii) the lack of flexibility and control over the environment dynamics makes it hard to generate



(a) a bird-eye view of the scenario

(b) a snapshot of GRF environment

```
builder.SetBallPosition(0.7, -0.28)
builder.SetTeam(Team.e_Left)
builder.AddPlayer(-1.0, 0.0, e_PlayerRole_GK)
builder.AddPlayer(0.7, 0.0, e_PlayerRole_CB)
builder.AddPlayer(0.7, -0.3, e_PlayerRole_CB)
builder.SetTeam(Team.e_Right)
builder.AddPlayer(-1.0, 0.0, e_PlayerRole_GK)
builder.AddPlayer(-0.75, 0.1, e_PlayerRole_CB)
```

(c) GRF's scenario program

```
behavior egoBehavior(player, destinationPoint):
        do Uniform(ShortPassTo(player), dribbleToAndShoot(destinationPoint))

behavior attackMidBehavior()
  try:
    do HoldPosition()
  interrupt when self.owns_ball is True:
    do AimGoalCornerAndShoot()
  interrupt when (distance to nearestOpponentPlayer(self)) < 5:
    do dribble_evasive_zigzag()

LeftGK
attack_midfielder = LeftAM on left_penalty_arc_region,
                facing north,
                with behavior attackMidBehavior()
ego = LeftLM ahead of attack_midfielder by Range(5, 10),
    facing toward attack_midfielder,
    with behavior egoBehavior(attack_midfielder, left_penaltyBox_center)

Ball ahead of ego by 0.1
RightRB left of ego by Range(3,5)
RightGK
```

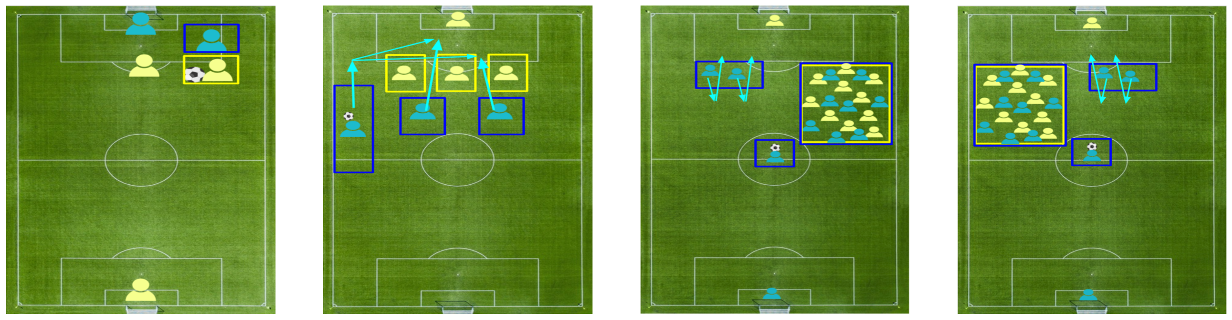
(d) Scenic program of generalized pass-and-shoot scenario with distribution over players' initial condition and behaviors

Figure 7.1: Comparison of Google Research Football simulator versus Scenic's models of pass-and-shoot scenarios

realistic evaluation scenarios to comprehensively test generalization in these complex RTS environments.

Using a scenario specification language whose syntax and semantics are carefully designed to intuitively model scenarios has the following benefits:

1. **Easily Model Interactive Environments on *User-demand* to Train and Test RL Agents:** The intuitive syntax and semantics, which abstracts away the implementation details and allows users to reason solely at high-level semantics, makes it easy to model complex spatial relations among multiple agents, their behaviors and conditions on how these behaviors should interact. It should be noted that, it requires a considerable amount of research and engineering effort to design and implement a formal scenario modeling language and its compiler from scratch.
2. **Program Stochastic Policies:** These programmed agents can serve two purposes: (i) allow developers to incorporate domain knowledge, e.g., generate demonstration data for offline training and (ii) provide performance baseline for trained RL agents.



(a) generalization test scenario for the scenario in Fig. 7.1a

(b) 3 vs 3 left midfielder crosses to either player in penalty box

(c) 11 vs 11 open player scenario

(d) mirrored Fig. 6.1 scenario

Figure 7.2: Examples of a new defense scenarios with specific assigned behaviors (a), a test scenario to assess generalization (b), and two full game scenarios (c,d) we used for training and testing. The RL team is yellow and the opponent, blue. The assigned opponent behaviors are highlighted with light blue arrows. Uniformly random distribution is assigned over a specific region for each player. These regions are highlighted boxes.

3. **Interpretability and Transparency**: The intuitive syntax and semantics make scenario programs interpretable and transparent. Therefore, users can reason about the difference/similarity of train and test environments by comparing their scenario programs.
4. **Reusability of Existing Scenarios**: The interpretability of scenario programs facilitates easy modification or re-use of existing Scenic programs, models, and behaviors to quickly model new scenarios. This facilitates building a community around designing and sharing scenario programs, by building upon each other's scenarios.

7.2 Methodology

A scenario is a Markov Decision Process (MDPs) [165] defined as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \rho_0)$, with $\mathcal{S}$ denoting the state space, $\mathcal{A}$ the action space, $p(s'|s, a)$ the transition dynamic, $r(s, a)$ the reward function, and ρ_0 the initial state distribution. Given the state and action spaces as defined by the GRF environment, a Scenic program defines (i) the initial state distribution, (ii) the transition dynamics (specifically players' behaviors), and (iii) the reward function. Hence, users can exercise extensive control over the environment with Scenic.

7.2.1 Modeling Initial State Distribution

Users can intuitively specify initial state distributions with Scenic's high-level syntax that resembles natural English. For example, refer to the full Scenic program in Fig. 7.1(d)

```
behavior receiveCrossAndShoot(destinationPoint):
    do MoveToPosition(destinationPoint)
    do HoldPosition() until self.owns_ball
    do dribbleToAndShoot(yellowPenaltyBox_centerPoint)
    do HoldPosition()

behavior crossToPlayers(list_of_players):
    destinationPoint = Point on region_in_penaltyBox
    do MoveToPosition(destinationPoint)
    do HighPassTo(Uniform(*list_of_players))
    do HoldPosition()
```

Figure 7.3: A snippet of a SCENIC program specifying behaviors for players Fig. 7.2b

which describes a more generalized version of GRF's Pass and Shoot scenario as visualized in Fig. 7.1(a,b). In line 12-22, the initial state distribution is specified. The SCENIC syntax for modeling spatial relations among players are highlighted in yellow. In addition, SCENIC supports about 20 different syntax to support modeling complex spatial relations [60]. Rather than having to hand-code positions for a concrete scenario as in the GRF's scenario 7.1(c), users can much more intuitively and concisely model a distribution of initial states. Here, `Left` represents the yellow team, `Right` the blue, and the two following abbreviated capital letters indicate the player role.

7.2.2 Modeling Transition Dynamics

One can flexibly modify transition dynamics of the environment by specifying the behaviors of non-RL players using SCENIC. Take the same example SCENIC program in Fig. 7.1(d) as above. Line 1-10 models two new behaviors. A behavior can invoke another behavior(s) with syntax `do`, succinctly modeling a behavior in a hierarchical manner. Users can assign distribution over behaviors as in line 2. The interactive conditions are specified using try/interrupt block as in line 5-10. Semantically, the behavior specified in the try block is executed by default. However, if any interrupt condition is satisfied, then the default behavior is paused and the behavior in the interrupt block is executed until completion and then the default behavior resumes. These interrupts can be nested with interrupt below has higher priority. In such case, the same semantics is consistently applied.

7.2.3 Rewards

SCENIC has a construct called `monitor`, which can be used to specify reward functions. The reward conditions in the `monitor` is checked at every simulation step and updates the reward accordingly.

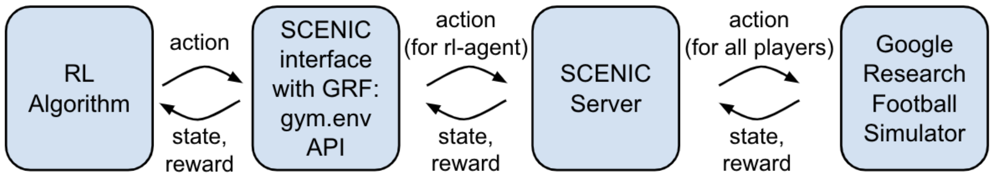


Figure 7.4: Interface Architecture between Scenic and an RL Simulator

7.2.4 Termination Conditions

Users can also specify termination conditions which are monitored at every simulation time step.

7.2.5 On Interfacing Scenic to a Simulator

Interfacing Scenic to other simulators is straight-forward. In fact, Scenic is already interfaced with five other simulators [62] in domains such as autonomous driving, aviation, and robotics. To interface Scenic with a simulator, one needs define the model, action, and behavior libraries. These libraries expedites modeling complex scenarios by helping users re-use the set of models, actions, and behaviors in the libraries, rather than having to write a scenario from scratch.

The *model* library defines the state space. It defines players with distribution over their initial state according to their roles and GRF's AI bot is assigned by default to all player behavior. These prior distribution over the initial state and behavior can be overwritten in the Scenic program. The model library also defines region objects such as goal and penalty box regions as well as directional objects in compass directions. The *action* library defines the action space as determined by the GRF simulator. These action space consists of movement actions in eight compass directions, long/short/high pass, shoot, slide, dribble, and sprint.

The *behavior* library consists of behaviors and helper functions that represent widely used basic skills in soccer. These behaviors include give-and-go, evasive zigzag dribble to avoid an opponent's ball interception, dribbling to a designated point and shooting, shooting towards the left or right corner of the goal, etc. Additionally, the behavior library also include useful helper functions such as identifying nearest opponent or teammate, whether there is an opponent near the running direction of a dribbler, etc.

7.2.6 Interface Architecture

Figure 7.4 shows an overview of our overall architecture. The architecture can be divided into two parts: i) RL interface, through which the RL algorithms interact with Scenic and ii) the Scenic Server, which executes a Scenic program and governs the simulation by interacting with the underlying simulator. We follow the widely used OpenAI Gym API [21] as our interface, which allows our interface to be used seamlessly with all the existing standard RL frameworks.

For each simulation/episode, the SCENIC server first samples an initial state from the SCENIC program to start a new scenario in the GRF simulator and updates its internal model of the world (e.g., player and ball positions). From then on, a round of communication occur between the RL algorithm and SCENIC server, with the RL interface at the middle. At each timestep, the gym interface takes in the action(s) for the RL agent and passes them to the SCENIC server. The SCENIC server in turn computes actions for all the remaining non-RL players—the players not controlled by the RL agent—and then executes all these actions (of both the RL and non-RL players) in the simulator. The SCENIC server then receives the observation and reward from the simulator, updates the internal world state, and then passes them back to the RL algorithm. This interaction goes on till any terminating conditions as specified in the scenario script is satisfied.

7.3 Experiment

In this section, we demonstrate four use cases of SCENIC in RL. First, we present and benchmark a set of 13 realistic mini-game scenarios encoded in SCENIC with a varying level of difficulty. Second, we test the generalization capabilities of the trained RL agents on unseen, yet intuitively similar scenarios. Next, we show how developers can "debug" their agents for failure scenarios of interest. At last, we show how probabilistic SCENIC policies can be used to generate offline data and endow domain knowledge into the learning process for faster training, which we believe to be very important for applying RL in practice.

7.3.1 Experimental Setup

We run PPO [150] on a single GPU machine (NVIDIA T4) with 16 parallel workers on Amazon AWS. Unless otherwise specified, all the PPO training are run for 5M timesteps and repeated for 10 different seeds. All the evaluation has been done for 10000 timesteps. For all the experiments, we use the stacked Super Mini Map representation for observations —a 4 x 72 x 96 binary matrix representing positions of players from both team, the ball, and the active player—and the scores as rewards, i.e., $+1$ when scoring a goal and -1 upon conceding, from [102]. Similar to the academy scenarios from [102], we also terminate a game when one of the following happens: either of the team scores, ball goes out of the field, or, the ball possession changes. For further details, including hyperparameters and network architecture, we refer readers to Appendix D.3.

7.3.2 Google Research Football Simulator

The Google Research Football (GRF) simulator [102] provides a realistic soccer environment to train and test RL agents. The setting, the rules, and the objective of the environment

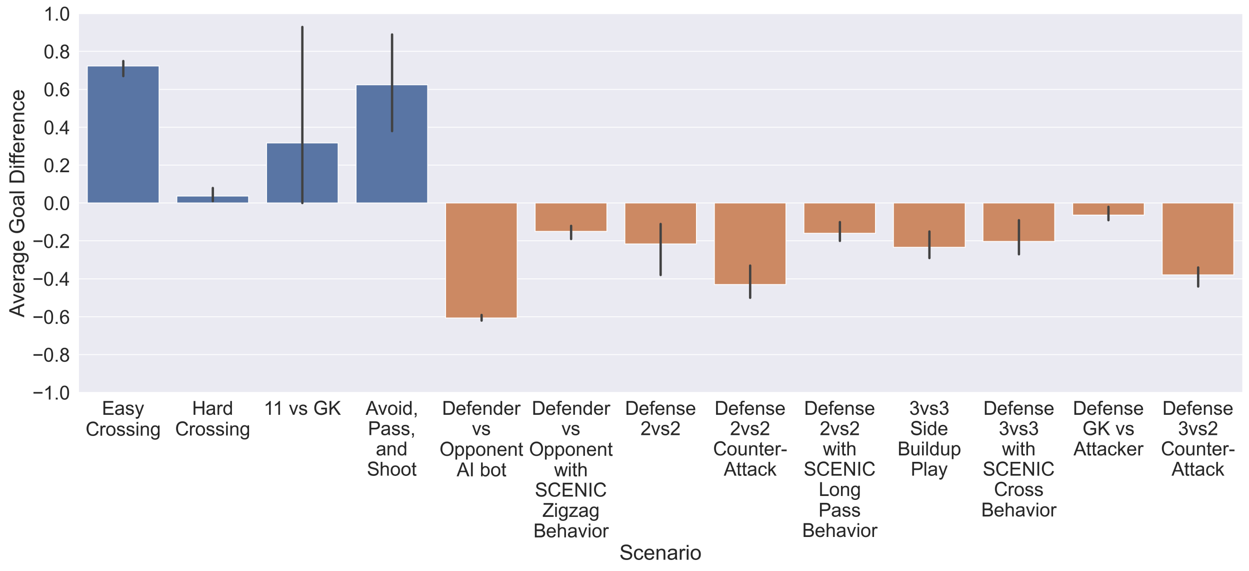


Figure 7.5: Average Goal Difference of PPO agents on our mini-game scenario benchmark. The error bars represent 95% bootstrapped confidence intervals

are the same as defined by Fédération Internationale de Football Association [53]. The environment setup is as the following. All the players on the field are controlled by (1) GRF's built-in, rule-based AI bots and (2) RL agents. The simulator dynamically determines which of the RL team players are to be controlled by RL agents based on their vicinity to the ball. GRF provides 11 offense scenarios to train and test RL agent performance and it provides trained RL agent checkpoints for a subset of its scenarios.

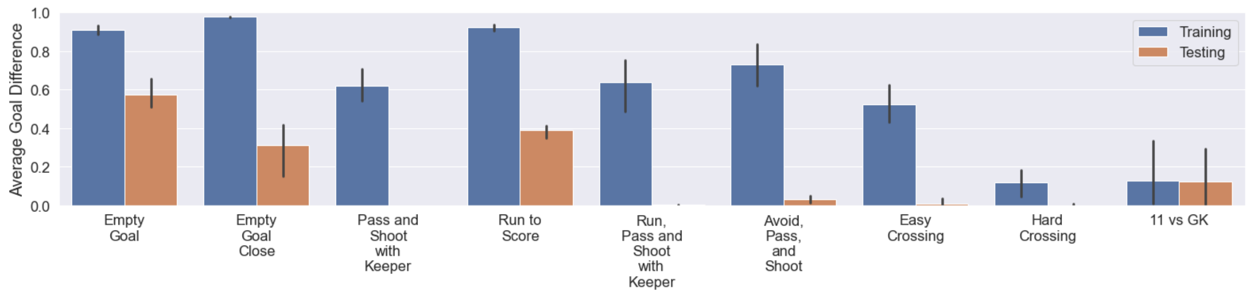
7.3.3 Mini-game Scenario Benchmark

Training an RL agent to solve a full soccer game involving 22 players is very challenging and may take days even with distributed algorithms. For example, [102] showed even the easy version of GRF's 11 vs 11 game cannot be solved with 50M samples. To allow researchers to iterate their ideas with a reasonable amount of time and compute, we present a set of 13 mini-game scenarios. All these scenarios are inspired from common situations occurring in real soccer games but involves fewer number of players to make them amenable to be faster training.

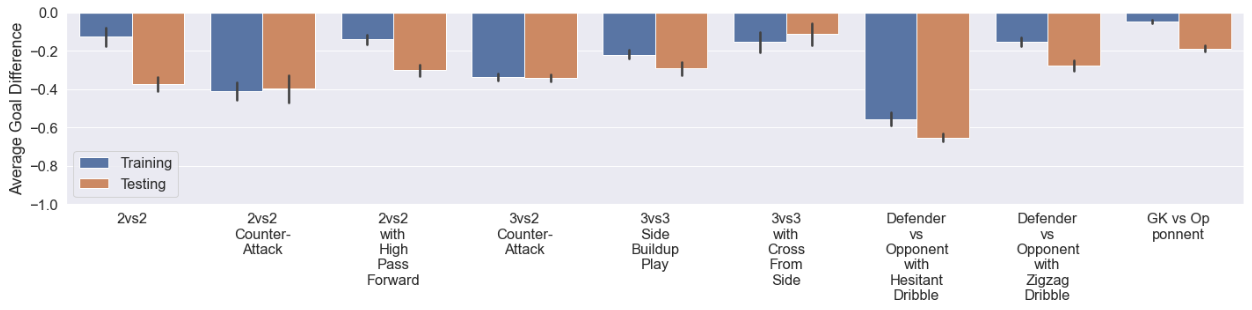
Nine of our mini-game scenarios are defense scenarios, which are nice complement to GRF's offense-only scenarios (refer to Sec. 7.3.2), along with four new offense scenarios. Most of these scenarios are initialized from a distribution, rather than fixed locations. By default all the opponent players are controlled by GRF's built-in AI bot (refer to Sec. 7.3.2). However, for the scenarios where the AI bot does not exhibit our desired behavior, we model the opponent behaviors using SCENIC. For example, in the 3vs3 cross scenario as shown in Fig. 7.2b, the opponent AI bots tried to pass the ball around instead of crossing. Therefore, we modelled and assigned behaviors such that the blue player on the leftmost side of the field would run up the field and cross the ball. Meanwhile, the two blue players in the center run into the penalty box area to receive the cross and shoot. These modelled behaviors are shown in Fig. 7.3.

We benchmark our mini-game scenarios by training agents with PPO. Figure 7.5 shows the average goal differences for all the scenarios. For these mini-game scenarios, we end the game if one of the teams score. Hence, the goal difference can range between -1 to +1. For the offense scenarios, a well trained agent is supposed to score consistently achieving an average goal difference close to +1. On the other hand, a well-trained agent should achieve a goal difference close to 0 for successfully defending the opponents in the defense scenarios. From the graph it can be seen that the scenarios offer a varied levels of difficulties. For example, PPO consistently achieves goal difference of around 0.5 for the Easy Crossing scenario, but barely learns anything for Hard Crossing. In case of the defense scenarios, the results also show a varied range of difficulty, GK vs Opponent scenario being be easiest.

7.3.4 Testing for Generalization



(a) Offense and select GRF academy scenarios



(b) Defense scenarios

Figure 7.6: Evaluation of PPO agents' generalization against varying initial conditions. For most of the academy and offense scenarios we observe a significant drop in performance. However, for several defense scenarios the difference in train and test scenarios is not that significant.

We provide scripts to test generalization of all of our 13 new benchmark scenarios along with 5 scenarios provided by GRF. We changed the distribution over the initial state while keeping the formation of players and their behaviors in each scenario intact. For example, for testing generalization of an RL agent trained in the Pass and Shoot scenario (Fig. 7.1a), we instantiated the yellow and the blue players on the symmetric right side of the field instead of the left and kept the other initial state distribution the same (Figure 7.2a).

Fig. 7.6 compares the trained agents' performance in training and test scenarios. As expected, we observe a noticeable drop of performance in most of the GRF's academy and offense scenarios (Fig. 7.6a). For example, the Pass and Shoot scenario (Figure 7.1a), which achieved around 0.6 in training, failed to generalize for the test scenario. However, for the defense scenarios, the drop in performance was not as noticeable. We conjecture that this distinction comes from the differences in the offense and defense training scenarios, where the defense scenarios tend to contain larger distribution over the initial state than those of the offense scenarios (refer to Appendix D). Consequently, larger variations of scenarios introduced during training may have contributed to better generalization for defense scenarios.

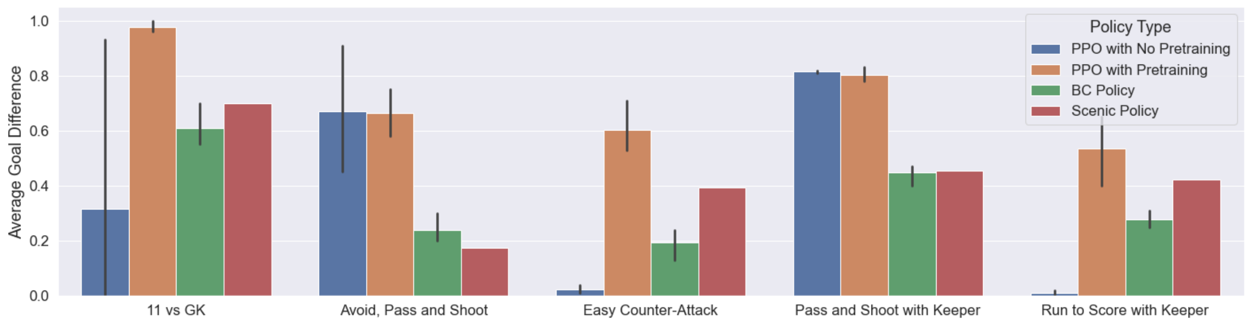


Figure 7.7: Performance comparison of PPO agents trained with and without any demonstration data from SCENIC, along with the performance of corresponding behavior-cloned and SCENIC policies. We see significantly better performance on three of the scenarios, while the rest two achieves comparable performance, highlighting the usefulness of the SCENIC policies.

7.3.5 Debugging Agents on 11v11 Failure Scenario

For this experiment, we evaluate and debug an RL checkpoint provided by GRF, which was trained on their 11 vs 11 easy stochastic scenario, i.e., easy version of their full-game scenario. This agent achieves an impressive average goal difference of 6.99 per full-game[1], scoring up to 14 goals in the training scenario during our experiments. We modelled a scenario, as

[1]Evaluated on 100K timesteps

visualized in Figure 6.1, to test the agent's ability to quickly perceive open teammates near the opponent goal to advance the ball forward and score—a crucial skill for soccer. When we assigned GRF's built-in AI bots to control the open players on the left side of the field, the players ran straight toward the ball, instead of taking advantage of the closeness to the opponent goal without being marked. Hence, we modelled a behavior for open players in Scenic so that they would stay close to the goal while abiding by the offside rule.

Although obvious to humans, the trained checkpoint performs poorly in this scenario with an average goal difference of 0.1. To 'debug' the agent, we then fine-tune the agent on a 'mirrored' scenario, as shown in Figure 7.2d, with PPO for 5M timesteps. The fine-tuned agent improved noticeably on the original scenario, achieving an average goal difference of 0.67. This showcases the usefulness of Scenic to easily model and generate scenarios of interest using one's domain knowledge, which may have been difficult with blackbox agents (e.g. built-in AI bots, or trained RL agents), to test and debug certain capabilities of an RL agent.

7.3.6 Facilitating Training with Probabilistic Scenic Policies

In the section, we show how RL practitioners can incorporate their domain knowledge by writing probabilistic Scenic policies for faster training. Using domain knowledge, we encoded RL policies in Scenic for five different scenarios, where the agent suffers to learn, and generated 8K samples of demonstration data for per scenario. To facilitate training on those scenarios, we first pre-train an agent via behavior cloning with the generated offline data and then fine-tune the agent using PPO for 5M timesteps. All the experiments were repeated for three different seeds. Figure 7.7 compares the training performance of these agents against the agents that were trained with PPO only. We notice that, even with such a low volume of demonstration data, we can train much better agents and can solve scenarios which were otherwise unsolved. The experimental results thus suggests, with stochastic Scenic policies we can generate rich quality demonstration data to substantially enhance training performance, which can be particularly useful in practice for environments like GRF which requires a heavy compute resource.

7.4 Bibliographic Notes

In literature, several techniques have been adopted to generate a rich variation of learning scenarios, primarily to promote or, ensure generalization. Techniques such as changing background with natural videos [199], introducing sticky actions [114] have been attempted, but are not robust enough. To ensure generalization, [107] and [154] generated training and testing scenarios by randomly sampling from different regions of parameter space. Similar to supervised learning, the use of separate train and test sets have also been adopted [128, 34, 35, 87], typically using techniques such as Procedural Content Generation [78], which has traditionally been used to automatically generate levels in video games. However, most of these focus on discrete domain, typically the dataset generation process is opaque, and it can be difficult to quantify or, reason about how different (or, similar) these train and test sets are, because the generation process often use random numbers to generate different configurations.

On the contrary, a few manually scripted scenario benchmarks are proposed with respect to a few RTS RL environments with limitations. For StarCraft [181], only two benchmark scenarios [177, 147] have been proposed. Both of these model different initial states but leave the behavior generation to either a learned RL agent or AI bots that are provided by the StarCraft environment, which are considered as blackbox agents. As a result, a sophisticated modeling and control over the behaviors of non-RL agents to create specific types of scenarios is not possible, severely restricting the diversity of the scenarios. For soccer domain, [161] presented one benchmark scenario on keepaway tactical scenario and later extended to more general half-field offense scenario [76] and provided a library of APIs relating to behaviors (e.g. mark player, defend goal) of players, which helps users to model scenarios. However, Scenic provides further benefits that are not covered in this work. Scenic provides high-level syntax and semantics to (i) easily write spatial relations for intutively modeling initial states, (ii) assign distributions over both initial states and behaviors to generate variations of environments for robust training and testing generalization, and (iii) specify priorities over interaction conditions over behaviors to model more sophisticated types of higher level behavior (refer to Sec. 2.1).

7.5 Chapter Summary

We introduced and demonstrated the benefits of adopting a scenario specification language to train RL agents and test their generalization capabilities in various realistic scenarios generated by Scenic programs, which succinctly capture distributions of initial states and behaviors. We also showcased modeling domain knowledge via stochastic Scenic policies by generating demonstration data to facilitate training in GRF, a complex real-time strategy environment. We believe that more advanced scenario modeling and generation support is necessary for the RL community to model dynamic physical interactions. Our work shows that a domain specific PPL like Scenic can.

Chapter 8

Personalized Human Training in Extended Reality

Human cyber physical systems (h-CPS) refers to cyber-physical systems that operate in concert with human operators [155]. To list a few examples, a fork truck maneuvered by a human driver and a drone whose trajectories are designated by a human are h-CPS. Using Scenic and VerifAI, we design and implement a personalized training algorithm to train humans interacting with h-CPS using augmented and virtual reality (AR/VR), or extended reality (XR) in short [90]. In our setting, a human wears an XR headset and fully controls the joints of a humanoid robot in XR to solve a task. The virtual humanoid tracks and mimics the human's body movements using the sensors embedded on the headset. The objective is to train each human a set of skills to control the humanoid in order to solve certain physical tasks. In particular, our algorithm aims to train psychomotor skills to humans, which consists of cognitive process and motor execution, and also assist humans to generalize the skills to similar tasks unseen during training.

To induce generalization, we use distributions of training tasks formally modeled as Scenic programs. In neuro-physiology, it is well-established that structured variability in training tasks promotes generalization of psychomotor skills [195, 43, 171, 15]. More intuitively, for example, to teach a sports player how to pass a ball to a running teammate, coaches may vary the teammate's running speeds and directions. These findings have been empirically demonstrated in adaptive motor skill training systems in reality [176, 112, 175, 174] and in XR [18, 162, 130]. Our system formally models this structured task distribution with Scenic and evaluate human performance with VerifAI.

A core contribution of this work is to develop a generic *personalized* training algorithm in XR, which can easily be adapted to train psychomotor skills for various types of h-CPS. Personalization in training is necessary because learning speeds and hand-eye coordinations vary among humans. Hence, the XR system needs to adapt the training to each individual. Our system is can easily be adapted for different domains because it solicit domain knowledge from the experts or instructors. In particular, key aspects of this domain knowledge include a set of skills to train and a corresponding set of training tasks which we formally model as

Scenic programs. Thus, each skill is associated with a Scenic program. Our algorithm models a human knowledge state and adapts training tasks in XR, accordingly. Once it selects a skill to train, the algorithm iteratively samples a task from the associated Scenic program and generates it in XR until the human reaches a mastery of the skill. We derive principles from learning sciences to algorithmically personalize training curriculum (i.e. the order of probabilistic programs to train with) and training speed (i.e. the number of tasks to sample per program to assist each user reach a skill mastery). In the rest of this section, we will interchangeably use the term, humans and users, which are equivalent.

8.1 Background: Bayesian Knowledge Tracing

We use Bayesian Knowledge Tracing (BKT) [198] to algorithmically model whether a human has mastered, or sufficiently learned, a skill. BKT has become the standard in education research for modeling a student's mastery of cognitive skills in domains such as algebra and chemistry. In fact, BKT is used in intelligent tutoring systems (ITS) [167] which are personalized tutoring algorithms developed in education research [142]. In ITS, for each cognitive skill (e.g. addition in algebra), there is a relevant question bank with solutions. ITS samples a question from the question bank and a student's performance is recorded as a boolean (i.e. correct or incorrect). Based on this boolean result, ITS uses BKT to algorithmically update the system's belief over the student's mastery of the skill. Similarly, in our setting, we have a probabilistic program in lieu of the cognitive question bank, modeling a distribution of psychomotor tasks. We use BKT as there is no literature on both adaptive and algorithmic approach to estimate psychomotor skill mastery. We believe that BKT potentially can estimate mastery because psychomotor skills also encompass cognitive process. Investigating the effectiveness of BKT is a key objective of this work.

Now, we explain the mechanism of BKT. A BKT model is tuned for each skill. Modeling BKT comprise of two parts: (i) a domain expert's analysis of training tasks designed to train a particular skill (e.g. a question bank related to addition in algebra) and (ii) then tuning BKT model's four parameters using their mental model of students: the student's initial probability of having mastered the skill from prior knowledge before training (prior), probability of the student mastering the previously not mastered skill after experiencing a training task (learn), probability to make a mistake when applying an already mastered skill (slip), and probability of correctly applying a skill that is not mastered yet (guess). For more detail, please refer to [198].

For context, BKT assumes a binary knowledge state, meaning that the student is either in the mastered or not mastered state with respect to each skill. It also assumes a binary-graded response from a student's attempt to solve a task (i.e. correct or incorrect). The underlying statistical architecture of BKT is a hidden Markov model with observable nodes representing the student's history of binary responses obs_t to a sequence of training tasks indexed with t, and hidden nodes representing students' latent knowledge state after experiencing t-th task. The mathematical definitions of these parameters and the Bayesian update rule is formulated below.

$$\begin{aligned}
\text{prior} &= P(L_0) \\
\text{learn} &= P(T) = P(L_{t+1} = 1 | L_t = 0) \\
\text{guess} &= P(G) = P(obs_t = 1 | L_t = 0) \\
\text{slip} &= P(S) = P(obs_t = 0 | L_t = 1)
\end{aligned}$$

Note that while $P(L_0)$ denotes the BKT's *prior* parameter, we also define $P(L_t)$ as the probability that the student has mastered the skill after experiencing t-th task. BKT updates $P(L_t)$ given an observed correct or incorrect response to calculate the posterior with:

$$P(L_t | obs_t = 1) = \frac{P(L_t)(1 - P(S))}{P(L_t)(1 - P(S)) + (1 - P(L_t))P(G)}$$

$$P(L_t | obs_t = 0) = \frac{P(L_t)P(S)}{P(L_t)P(S) + (1 - P(L_t))(1 - P(G))}$$

The updated prior for the following time step, which incorporates the probability of learning, is defined by:

$$P(L_{t+1}) = P(L_t | obs_t) + (1 - P(L_t | obs_t))P(T)$$

8.2 Methodology

In this section, we present the design of our intelligent, personalized XR training system to support users learn and generalize relevant psychomotor skills to engage in diverse XR activities of interest. Our overall system design is visualized in Fig. 8.1. Because different XR domains, or applications, often do not readily have offline training data to leverage any data-driven machine learning techniques, we utilize domain expertise to compensate. Although our design methodology is not particular to any specific XR domain, for ease of explanation, we will use our interaction with VR esports experts as a running example.

8.2.1 Solicit and Implement Domain Knowledge

Once an XR domain for training is determined, we recruit the domain experts to gather following information, which serves inputs to our system as shown in the leftmost part of Fig. 8.1: (i) a set of skills to train, (ii) prerequisite relations among the skills, and (iii) corresponding distributions of training and evaluation tasks for each skill with task evaluation metrics, and (iv) BKT parameters for each skill (refer to Sec. 8.1).

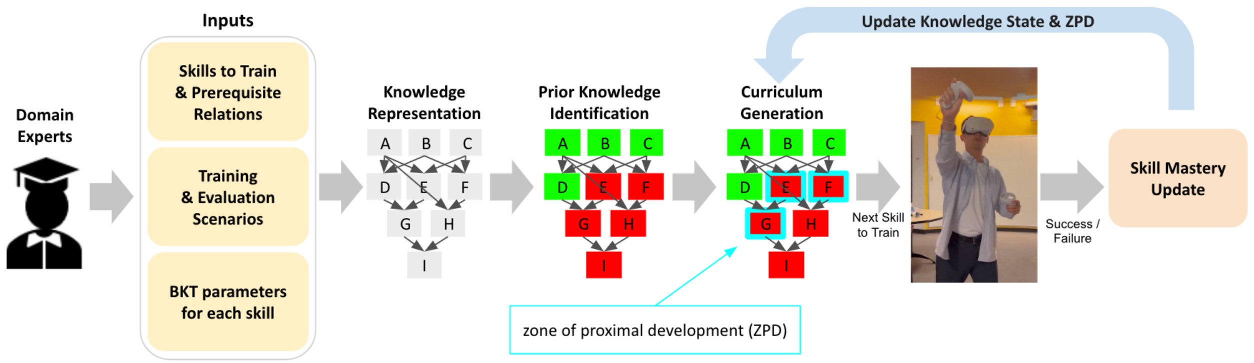

Inputs
Domain Experts
Skills to Train & Prerequisite Relations
Training & Evaluation Scenarios
BKT parameters for each skill
Knowledge Representation
Prior Knowledge Identification
Curriculum Generation
Update Knowledge State & ZPD
Next Skill to Train
Success / Failure
Skill Mastery Update
zone of proximal development (ZPD)
A B C D E F G H I

Figure 8.1: Overview of our personalization approach

Identifying Skills to Train and the Inter-relations Among the Skills

First, we conduct to a joint meeting with experts to identify and represent the training skills and their interrelation as a knowledge graph shown in Fig. 8.2. To facilitate the discussion, we used a shared PowerPoint slide. We ask the experts to first brainstorm which psychomotor skills are fundamental to engage in the chosen XR domain, and type the names of the skills on the slide so others can see. Once a sufficient number of skills are written down, we ask the experts to discuss and reach a consensus on which skill to train. Once the set of training skills are determined, on the shared slide, we create a set of blocks, each with a skill name inscribed as shown in Fig. 8.2 to facilitate the discussion on pre-requisite relations among skills. We ask the experts to first identify blocks (i.e. skills) that do not have any pre-requisite skills. Once identified, we re-arrange the identified blocks (e.g. T, GR, SP) to form a top level in the shared slide as shown in Fig. 8.2. Then, we inquire the experts to place the blocks, which immediately require the skills at the top level, right underneath the top level and indicate the pre-requisite relation with directed arrows, where the skill pointed at requires the skill pointed from. We iterate this process until all blocks are consumed, forming a directed, acyclic, pre-order graph, i.e. in short, knowledge graph, for example, as shown in Fig. 8.2.

Designing Task Distributions & BKT Tuning

Next, we solicit further domain knowledge on training and evaluation tasks and related BKT parameters for the identified skills. In the video call, we share a shared Figma [50] document for the experts to draw out the imagined training and evaluation tasks with variations. For each skill, we ask the experts to collectively discuss and determine the tasks and their evaluation metric by drawing them in the shared document. After they complete modeling a training task distribution for each skill, we ask the experts to tune BKT parameters for the skill based on their knowledge of the task distribution. For tuning, we specifically ask the following questions in Likert 5-point scale [108], assuming that the user already mastered the prerequisite skills.

1. There is a high chance a novice user will learn the skill after a single training exercise. (learn)
2. A user is likely to solve the task in a training task without having mastered the necessary skill. (guess)
3. Considering the complexity of the maneuvers that a novice user has to make to solve for the training task, a user is likely to make a mistake and fail to solve a task in this task even if they had already mastered the necessary skills. (slip)

These BKT parameters are probabilities ranging [0,1]. However, the Likert 5-point scale may not directly map to [0,1] (e.g. BKT's *learn* probability of 4 out of 5 points may not necessarily map to 0.8 in probability). Hence, we further ask experts to provide their estimate on the *consecutive* number of training tasks, N, a user needs to solve to reach a mastery. This practice of enlisting experts to help hand set BKT parameters based on expected skill learning trajectories, is not unique to our work. In the first few years of operation, this was the practice established by the Cognitive Tutor [142], an intelligent tutoring system, for setting their skill parameter values.

Implement Solicited Domain Expert Knowledge

We represent the interrelation among skills as a knowledge graph as shown in Fig. 8.1 under "knowledge representation," whose nodes are skills and directed edges, pre-requisite relations. Per skill, the associated training and evaluation task distributions are modeled as two distinct probabilistic programs, respectively. The corresponding task evaluation metrics are programmed with Python [145]. Finally, we implement a BKT model for each skill. Traditionally, the standard use of BKT is that skill mastery is reached if BKT's prediction (in probability) is greater than 0.99. Regarding the "prior" parameter, we conservatively uniformly set it to 0.05 across all skills since we do not have data a priori for estimation. For the remaining three BKT parameters, for each skill, we find a mapping from the experts' Likert 5-point scale to probability of [0,1] such that correctly solving N consecutive tasks result in BKT's output greater than 0.99. Recall that this number N varies across skills and are provided by the experts (refer Sec. 8.2.1).

8.2.2 Personalized Curriculum Generation

Prior Knowledge Identification

To personalize the curriculum, the system needs to first identify the user's prior knowledge of skills. The user's knowledge state, as visualized in Fig. 8.2, is defined as a *colored* knowledge graph, whose binary colors represents mastered (green) or not mastered skill (red), respectively. This binary knowledge state assumption derives from BKT (refer to Sec. 8.1). Our goal is to efficiently color the uncolored knowledge graph as visualized in Fig. 8.1, under "knowledge representation," to fully colored knowledge state shown under "prior knowledge identification."

Without having to test a user with every skill, the system efficiently select a subset of the skills to approximate prior knowledge leveraging the interrelation among skills. Once a skill is selected for an evaluation, the system sample and generate evaluation tasks from its corresponding probabilistic program for N number of tasks that experts provided to reach mastery for the skill (refer to Sec. 8.2.1). If the user solves all the N tasks, indicating mastery, then the system colors the node and its pre-requisite nodes in the knowledge graph to be green and update the nodes' associated BKT models' *prior* parameter to be above 0.99. However, if the user has not master the skill, then the system colors the node and its post-requisite nodes as red.

To efficiently sample the evaluation skills, the system uses the algorithm in Equation (1). For each *uncolored* node in the knowledge graph, this algorithm computes the time saved from evaluating a skill, s. Suppose the user has already mastered the skill, s, then the system saves the time to evaluate its pre-requisite skills, which we denote t_s^+ and computed by summing the task completion times of pre-requisite skills. Similarly, t_s^- sums the task completion times of post-requisite skills for the case when the user has not mastered the skill, s. Hence, for each uncolored node, s, the worst saved time is $\min(t_s^+, t_s^-)$. The system samples for the uncolored node, which maximizes the worst saved time, for a time-efficient prior knowledge identification. We mathematically formulate the algorithm for sampling skill for prior knowledge identification in Equation (1).

$$s^* = \underset{\text{s is uncolored}}{\arg\max} \; \min(t_s^+, t_s^-) \tag{8.1}$$

Adaptive Curriculum Generation

Zone of proximal development (ZPD) is a concept from psychology, which we adopt to generate a personalized curriculum. ZPD defines the "boundary zone" of human knowledge, which defines the zone that is not learned yet but has close relation with those already learned. Previous literature shows that, with activities selected from ZPD, students can learn on their own with little guidance from instructors [105, 111], and feel more engaged in learning[38].

As highlighted in light blue in Fig. 8.1, under “Curriculum Generation,” we define the ZPD to be a *set of red color nodes* that are either one edge away from the green nodes or red nodes with no prerequisite skill. From the ZPD set, the system selects for the *next* skill to train, which has the minimum number of prerequisites. If there are more than one such node, the system chooses the one with a shorter time constraint for its training task. The system uses these heuristics to expedite the training. Once the user masters the selected skill, then the system updates the knowledge state and ZPD accordingly and sample another skill to train. The system repeats this procedure until either all skills are mastered or training time expires.

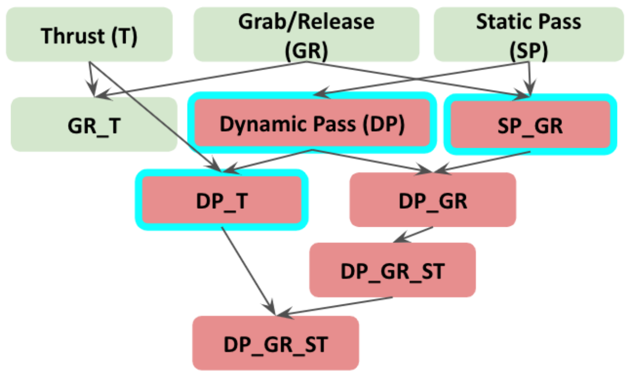


Figure 8.2: Our system represents a knowledge state as a colored, acyclic, directed, pre-order graph as visualized in this figure. Each node represents a skill. The directed edges encode prerequisite relations. The color represents mastery (green: mastered, red: not mastered). The zone of proximal development (ZPD) highlighted in light blue is a set of not mastered skills that are *in proximity* to mastered ones.

8.2.3 Personalized Training Speed

Once a skill is selected by our curriculum generator (refer to Sec. 8.2.2), the system searches for the skill's probabilistic program, modeling training task distribution, and its task evaluation metric. The system inputs these two to VerifAI toolkit. VerifAI samples a training task from the probabilistic program, generate it XR, and evaluates the user (i.e. correctly or incorrectly solved the task). This binary response, or evaluation result, is used to update the associated BKT prediction. If the BKT does not predict mastery, then the knowledge state and the ZPD set remains the same. Hence, more training tasks are sampled and generated until BKT predicts mastery. This way, our system adapts the number of tasks, or practices, for each user to reach mastery per skill.

8.3 Experiment

In this section, we conduct a between-subjects study to evaluate our system which uses BKT's predictions to personalize the curriculum (i.e. the order of skills to train) and the training speed (i.e. the number of practices, or tasks, per skill to sample from each probabilistic program modeled in Scenic to assist each user achieve mastery). The control condition uses self-assessment in lieu of BKT to control the curriculum and training speed. We use self-assessment for comparison as it is, to the best of our knowledge, currently the only *adaptive and scalable* methodology to predict psychomotor skill mastery with respect to task distributions (refer to Sec. 8.4).

The three hypotheses of our study are: **(H1)** our system is more effective than the self-guided learning of psychomotor skills in terms of learning gains and subjective task load, **(H2)** BKT is more accurate design component in predicting psychomotor skill mastery with respect a task distribution, to personalize training speeds, and **(H3)** our system's personalized curriculum based on BKT predictions is more effective than the control's in terms of user experience and training efficiency.

8.3.1 Example Application Domain: Esports

Esports is an interesting application domain which require skills that *encapsulate diverse characteristics of psychomotor skills in general.* It requires both fine (e.g. hand, feet) and gross (e.g. arm, legs, waist) movements, while involving careful tactical cognitive planning. Also, it involves physical coordination with other dynamic virtual agent(s). For these reasons, we select Echo Arena, a zero gravity frisbee VR esports, as our example application domain to conduct our study. We reconstruct Echo Arena in Unity [73] and interface Scenic to model and generate the desired training and evaluation scenarios in VR.

8.3.2 Experts/Instructors Recruitment

We recruited four professional Echo Arena esports players via direct messaging on Discord [82]. They provide us with necessary inputs (refer Sec. 8.2) to our training system through 2 hours of joint video call. Each professional is paid $50 for their time and inputs. These professionals have achieved the top 10 in ranking over the last few years in the VR Master League [104], which hosts the largest annual Echo Arena tournament. For context, in the most recent tournament in 2022, nearly 8,000 people around the world joined the competition [104]. These four experts also has experience in coaching novice or amateur Echo Arena players.

8.3.3 Participants

We recruited participants through university online forums and mailing lists from a community of VR users. We receive 25 responses of subjects with prerequisite dynamic VR game experience. Out of the 25 respondents, we exclude 7 subjects according to our three *pre-determined* exclusion criteria: 1) exhibiting motion sickness, 2) too much skill expertise (no opportunity for learning), and 3) extreme lack of hand-eye coordination (unlikely to master any skill during our short training session). The accepted 18 participants' ages range from 19 - 25 years, with 4 females and 14 males. Eligibility criteria and a summary of participants' backgrounds are listed in the supplemental material. Each participant is financially compensated with $40 gift card for their 2 hours of participation. For the participants who are excluded according to our pre-determined criteria, they are compensated for the time they participate at $20 per hour rate.

8.3.4 Procedure

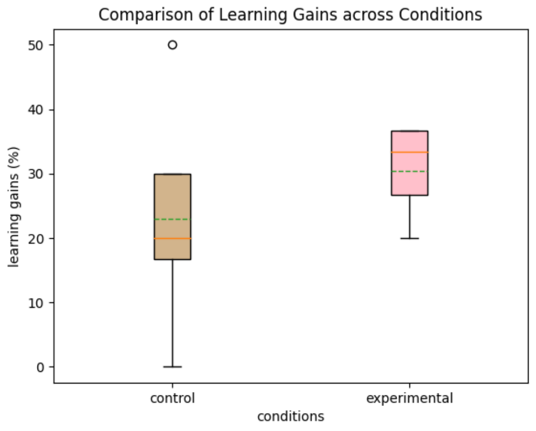
We conduct an IRB approved between subjects experiment to avoid learning and fatigue effects. We randomly divide the accepted 18 participants into two disjoint groups, i.e. the control and the experimental groups, with 9 participants in each condition. The study is conducted individually, not in groups. The study consists of the following sessions: basic tutorial (5 min), pre-test (15 min), advanced tutorial (10 min), training (25 min), post-test (15 min), and exit questionnaire (5 min), with 10 min breaks in between sessions including the half way through the training session. The details of each session is explained in the supplement. We train and evaluate 10 different skills provided by our recruited experts. The pre/post tests examine each of the 10 skills with a variable number of consecutive tasks, which are sampled from its corresponding evaluation task distribution, modeled as a Scenic program, and sequentially generated in XR. Recall when tuning BKT parameters for each skill (refer to Sec. 8.2.1), we ask the recruited experts, per skill, how many *consecutive* tasks a participant need to solve to demonstrate mastery. We use this information to set the number of evaluation tasks per skill (refer to our supplement for these numbers).

Both conditions follow the exact same procedure as above, *except for the training session.* During training, the experimental condition is trained with our system which automatically adapts the training contents (i.e. the curriculum and the training speed) to each user using BKT's predictions. In contrast, the control condition manually adapts the training contents using self-assessment of skill mastery after completing each task. In addition, the control is provided with a default curriculum designed by our recruited experts, and are given the freedom to modify it as they see fit. *The control condition is informed that the default curriculum is designed by experts.* After completing each task, the control condition is asked in XR for their (i) binary self-assessment (i.e. mastered/ not mastered) on the mastery of the current skill, and (ii) whether to transition to another skill. Either until the participant decides to transition (control) or BKT predicts mastery (experimental), tasks are iteratively sampled from associated probabilistic program and generated in XR. Because the periodic self-assessment takes up a small portion of training time, we also ask the experimental condition to also self-assess after each task, even though it is not used. During training we collect the following data. After subjects completes each task, we record the task name, binary task evaluation result (i.e. correct/ incorrect), BKT's prediction, and binary self-assessment prediction, and time when the data are collected.

8.3.5 Measurements

Learning Gains A learning gain for a participant is computed by one's score improvement (i.e. post test - pre test scores), where the pre and post test scores are computed in the following way: $\sum_{i=1}^{K}$(# of correctly solved tasks for skill i) / (total # of tasks used to evaluate skill i), where K is the number of skills to train. In this study, K=10, and each skill is evaluated with a variable number of tasks (refer Sec. 8.3.4).

Statistical Significance Test We use Mann-Whitney's U test using Python Scipy's stats package [86] for all the statistical significance tests reported in the Results Section. We choose this test because the sample size is too limited to expect normal distributions to hold for unpaired t-test.



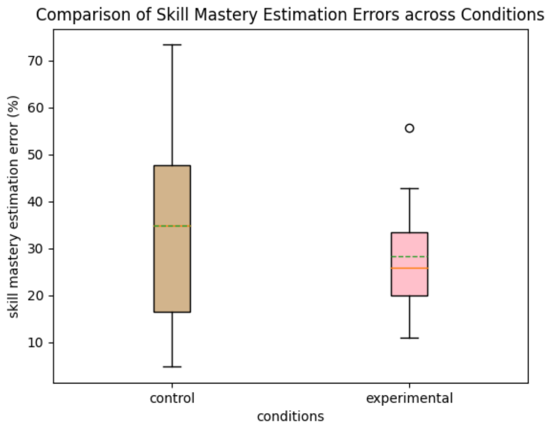


Figure 8.3: The left box plots compare the average learning gains between the two conditions. The right box plots compare the BKT and the self-assessment's average errors in predicting psychomotor skill mastery. The green dotted line in the box plot represents the average and the orange line, the median.

User Experience Evaluation We use the NASA task load index (TLX) [75] to measure a subject's subjective mental workload for the skills we train, before and after the training session. To measure the improvement in the subjective task load, we compute (TLX score after training - TLX score before training). Furthermore, we conduct exit interviews to qualitatively evaluate subjects' training experience for both conditions. We ask for any positive or negative experiences with the training. And, we inquire following three Likert 5-Point scale questions to evaluate user experience with the training curriculum.

1. The training session was engaging.
2. The training session was incrementally challenging.
3. The training has helped me learn new skills in virtual reality.

A table listing out the 5 point scale and their meanings, i.e. strongly disagree, disagree, neutral, agree, strongly agree) was provided underneath each statement.

Approximating Mastered Skills in Pre/Post Tests In the evaluation session (i.e. pre/ post tests), a user is examined with a variable, consecutive number of tasks per skill (refer to Sec 8.3.4). We approximate that the user has mastered a skill if one can solve all the consecutive tasks for the skill in the evaluation.

Skill Mastery Prediction Error This error is computed using the difference between the expected and the actual post test scores for the skills that are predicted to be mastered by either BKT or self-assessment, i.e. $M - \sum_{i=1}^{M}(\#$ of correctly solved tasks for skill i) / ($\#$ of tasks used to evaluate skill i) where M represents both the number of mastered skills and the expected score for each participant.

8.3.6 Results

Effectiveness in Learning Gains Prior to comparing the learning gains between the two conditions, we check whether there is any imbalance in the prior skills between the two conditions. The difference in the distributions of the pre-test scores is not statistically significant (p-value < 0.05). Regarding learning gains, the experimental group outperform the control group on average with statistical significance (p-value < 0.05) as shown in Fig. 8.3 with an effect size of 0.41. On average, the control group improves $22.96 \pm 12.90\%$ in learning gains, whereas the experimental group improves $30.37 \pm 5.97\%$. We observe that the standard deviation of the experimental condition is reduced by 53.7% than the control's.
User Experience Despite the higher average learning gains, the experimental condition does not result in lower subject task load after training than the control. Recall that lower TLX score is preferable because it means the subjective task load has decreased. The experimental condition shows a mild average decrease in NASA TLX scores by $6.56 \pm 16.00\%$, while the control exhibits a medium average decrease by $17.56 \pm 11.77\%$. However, the difference in the distributions of TLX score improvements is not statistically significant (p-value 0.07).

After the post tests, we equally ask both conditions for their user experience with training curriculum, using our Likert 5-point scale questionnaire related to engaging, incrementally difficult, and helpfulness in learning new skills. Both conditions positively rate their training experience as plotted in Fig. 8.5, averaging approximately 4.5 out 5 points for all three aspects. Mann-Whitney U test show that the differences in distributions across conditions for engagement, incremental difficulty, and helpfulness are not statistically significant, reporting p-values of 0.86, 0.43, and 0.34, respectively.

During the verbal interview, we also ask both conditions for any negative experiences with the training in general. While the control condition does not share any negative feedback, four out of nine participants in the experimental condition (we denote participants as E1-E9) report negative experiences particularly with the training speed. Some participants share frustration from too many assigned practices for a specific skill:*"I got frustrated towards the end because I was stuck in a task"* (E3) and *"getting stuck in a task was a bit frustrating in the beginning, but frustration went down as I saw myself improving"* (E5). On the contrary, some report premature transitions: *"sometimes, the training algorithm transitioned you a bit earlier than you expected"* (E6) and *"during the training, I thought I still needed some more practice, but during evaluation I actually performed better than I expected"* (E1).
Skill Mastery Estimation We compare the skill mastery prediction errors between the two conditions. Our results show that BKT has lower average prediction error than self-assessment, as visualized in Fig. 8.3 (right). BKT overestimates participants' skill mastery by $28.21 \pm 13.06\%$, whereas the control overestimate by $34.81 \pm 23.67\%$. However, these two distributions of prediction errors is not statistically significant (p-value 0.46). We compute the Pearson correlation [56] between participants' actual scores to the corresponding expected scores. The experimental condition has a noticeably higher correlation coefficient of 0.96 (p-value < 0.01) than the self assessment's 0.59 (p-value 0.09).
Curriculum Generation Our results show that none of the control participant alters the

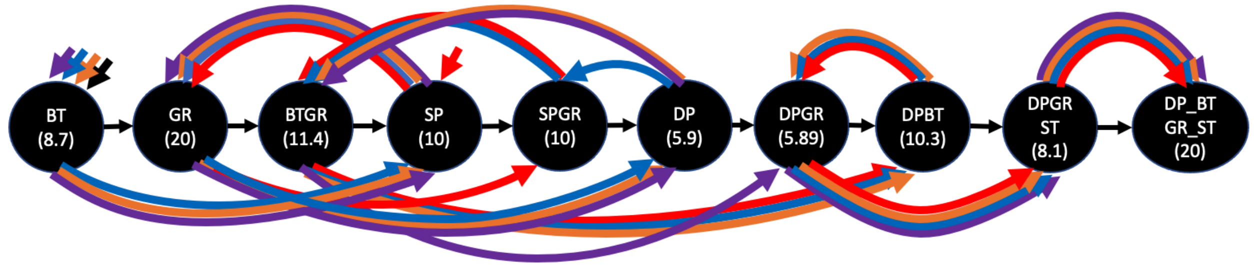


Figure 8.4: A comparison in curricula between the control and the experimental conditions. The black circles represent skills to train and the arrows visualize different training curricula (i.e. the order of skills). The control condition adhered to a single curriculum shown in black arrows. In contrast, four different curricula emerged in the experimental conditions as highlighted with purple, blue, orange, and red arrows.

default expert-designed curriculum, even though they have the freedom to do so as they see fit. On the contrary, our system generates four unique personalized curricula according to four different categories of identified prior knowledge in the experimental condition. The comparison between the control's uniform versus the experimental condition's personalized curricula is visualized in Fig. 8.4. In contrast to the control's fixed curriculum (highlighted in black arrows), our training system skips over skills (e.g. T, DP, SP_GR, DP_T) that participants already master to focus the training on the skills yet mastered.

Training Efficiency In terms of the average number of skills that are predicted to be mastered during training, both conditions achieve similar outcomes. In the control, participants self-assess to have mastered 7.78 ± 1.98 skills on average during the training session. In the experimental condition, BKT predicts that participants master on average 7.56 ± 2.51 skills in training. The difference between the two distributions is not statistically significant (p-value < 0.05). Furthermore, for the experimental condition, the average time to complete prior knowledge identification is 6.49 ± 0.74 minutes, which is much more efficient than testing each participant with all the skills as in the pre-test that takes 15 minutes. In this context, our results show that the experimental condition spends nearly $1/3$ of training tasks on skill already mastered than the control, and efficiently focuses the training on not mastered skills. The control spends on average 16.67% of training on skills they already master (i.e. on average 11.67 ± 10.21 tasks out of 66.33 ± 14.53 total average number of tasks completed in training). In comparison, the experimental condition spends 6.57% of training on the skills they already master (i.e. on average 4.16 ± 4.53 tasks out of 63.33 ± 8.77 total average completed tasks). For fairness, we report time in terms of tasks, not minutes, because each skill's training task requires different average task completion time, ranging from 6 to 20 seconds. As a result, a single task for a skill may worth 3 tasks for another skill in terms of task completion time.

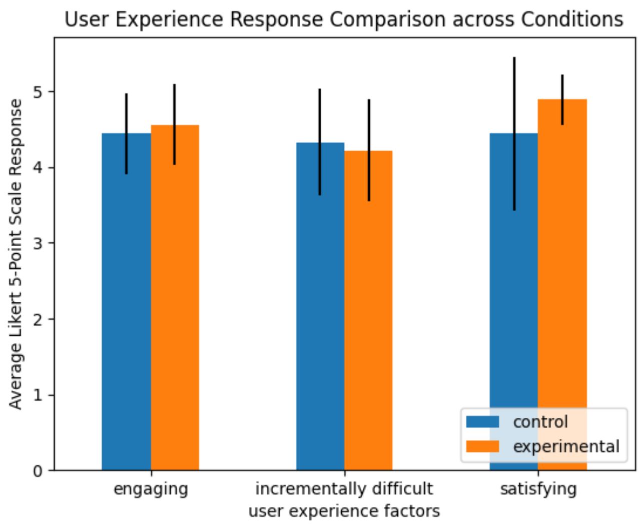


Figure 8.5: The bar plots compare the user experience on the training curriculum with respect to engaging, incrementally difficult, and helpfulness in learning new skills across the two conditions.

8.3.7 Discussion

In this section, we analyze our results in relation to our hypotheses (H1-H3) as stated in Sec. 8.3. Based on observations of our study, we suggest directions to improve observed shortcomings of BKT, which serves as the backbone of our system, in Sec. 8.3.7. Finally, we share potential challenges as others may author our system in Sec. 8.3.7.

System Analysis

We analyze our study results in relation to our three hypotheses (H1-H3) (refer to Sec. 8.3).
H1: Effectiveness of the System As visualized in Fig. 8.3 (left), the users trained with our system exhibits higher average learning gains than those train relying on self-assessment (p-value < 0.05), but with a mild effect size of 0.41. Much more noticeable is the consistent learning outcome with the experimental condition, whose standard deviation of learning gains ($\pm 12.90\%$) is 53.7% lower than the control condition's ($\pm 5.97\%$). This shows that our system considerably lowers fluctuations in the learning outcomes than the control, reducing the chance of users falling behind. This is a necessary characteristic to support a wider user base to learn relevant skills to engage in diverse XR activities of interest.

Counter-intuitively, despite higher average learning gains, the experimental condition does not result in lower average subjective task load than than control after training. Recall that it is desirable to lower subjective task load (i.e. lower NASA TLX score) through training. The control condition decreases in the NASA TLX score by $17.56 \pm 11.77\%$ on average, whereas the experimental only decreases by $6.56 \pm 16.00\%$. Although the difference is not statistically significant, we observe the average leaning more favorably towards the control.

As pointed out in the verbal interview, this potential cause of this outcome derives from our use of BKT to control training speeds. While the control group report no negative experience with training, nearly half of the experimental condition do, specifically concerning our system's training speed. The participants in the experimental condition (E3,E5) repeatedly use the word "stuck" to share their frustration from *not being able to stop excessive training* on a particular skill. Also, E1 and E6 complain that the system *prematurely transition* them to training new skills even though they do not feel prepared to move on. This discrepancy in the learning gains and subjective task load has been observed in academic learning setting (e.g. [166]) as well, where the condition achieving the highest objective learning gains also counter-intuitively result in the highest subjective task load, due to the training system's limited interaction with users.

H2: Accuracy of BKT vs. Self-Assessment The accuracy of the BKT and the self-assessment directly affects the training speed. Recall, when training a skill, tasks are iteratively sampled from associated probabilistic program and generated in XR until either BKT predicts mastery (experimental) or the participant self-assesses to transition to another skill (control). The results show that the BKT's skill mastery prediction error ($28.21 \pm 13.06\%$) is lower on average than the self-assessment's ($34.81 \pm 23.67\%$), but with no statistically significant difference. However, *BKT significantly lowers fluctuations in mastery prediction errors* as visualized in Fig. 8.3 (right). Due considerably lower variations in its prediction errors, BKT's predictions are 37% more highly correlated to skill mastery than the self-assessment's. We also note that BKT's high correlation does not derive from consistent underestimation of skill mastery, thereby providing over-practices to train a fewer number of skills. The control and the experimental conditions completes on average 7.78 ± 1.98 and 7.56 ± 2.51 skills, respectively, out of 10 skills during 25 minutes of training session, with no statistical difference. These findings explain why the experimental condition exhibits more consistent learning outcomes and achieve higher average learning gains.

These disparity in the consistency of predictions becomes clearer as we contrast the processes involved in the two methods. Recall, by our definition, a user who achieves mastery of a skill can correctly solve all tasks in its associated training task distribution. For self-assessment, this means that each user needs to: first, accurately approximate the task distribution from experiencing sampled tasks. Second, the user needs to accurately assess confidence in solving tasks *with respect to one's estimated mental model of the the task distribution.* This approximation of the task distribution likely becomes challenging as task complexity increases. Furthermore, it is well-established [49, 69] that self-assessment is inaccurate during learning phase.

On the contrary, for modeling BKT per skill, domain experts tunes BKT parameters (refer to Sec. 8.1) using their accurate understanding of the task distribution *that they designed for training the skill* and mental models of novice students' learning processes. We conjecture that the observed BKT's prediction errors derive from its insufficient parametrization of learning process, since BKT is traditionally developed for purely cognitive skills (e.g. algebra) which does not involve physical movements. Furthermore, the errors may also stem from the domain experts' inaccuracy or bias in their mental models of students or in expressing their knowledge as BKT parameters.

H3: Effectiveness of Personalized Curriculum We observe a stark difference in the generated curricula between the two conditions as visualized in Fig. 8.4. Our system generates four unique personalized curricula according to four different categories of prior knowledge identified in the experimental condition. In contrast, the control uniformly adheres to a single expert-designed curriculum. We compare both conditions' user experience on the curriculum with our customized Likert 5-Point questionnaire. As shown in Fig. 8.5, both conditions positively report that the training is engaging, incrementally difficult, and helpful for learning new skills, averaging around 4.5 out of 5 across all three aspects, with no statistically significant difference. This shows that the effects of our system's curricula on the user experience with respect to the three aspects are not statistically different from the expert-designed curriculum.

With the combined effect of personalized curricula and training speeds, the study results show that our system has higher training efficiency. The prior identification phase of our system takes on average 6.49 ± 0.74 minutes, which is 56.7% more efficient than the pre-test that takes 15 minutes to test every skill. Instead, our system's prior identification algorithm selectively tests a subset of skills using their pre-requisite relations. Following prior knowledge identification, our system spends only about 1/3 of time (measured with the number of tasks) on the skills they already master compare to the control's, and efficiently focus on training not mastered skills. Our system efficiently skips over mastered skills and more consistently predict skill mastery to adequately transition each user to train for skills that are not mastered yet. These findings also explain the experimental condition's higher average learning gains than the control's.

Our Suggestions to Improve BKT

The key insight to take away from our work is that BKT is more reliable design component than self-assessment for psychomotor skill mastery prediction. This insight could be incorporated to designing other adaptive training system which uses distributions of training tasks for a generalizable training. In our system, we demonstrate the two use cases of BKT to personalize the training curriculum and speed. However, our study results also reveals BKT's shortcomings concerning (a) the inaccuracy of BKT itself and (b) inadequate use of BKT for personalizing training contents. We suggest directions to improve BKT's accuracy and its usage in this section.

To improve its accuracy, the canonical formulation of BKT needs to be extended with more variables relating to influential physical factors such as fatigue. During pre- and post-test phase of our study, we frequently observe participants failing to solve tasks in post-tests, which they were able to solve correctly before in the pre-test. We conjecture that the accumulation of visual fatigue from exposure to VR and physical fatigue from exertions may have induced the outcomes we observed. However, further investigation is necessary.

Our study reveals two issues regarding our system's use of BKT's predictions to personalize training contents: (a) premature transition before the users feel confident with a skill and

(b) frustration from excessive practices of a same skill. To prevent premature transitions, it may be appropriate to probe and incorporate the user's self-assessment of the current skill *after BKT predicts skill mastery.* If the user is not confident, then more tasks should be sampled and generated in XR until the user is confident. This way, we can align the BKT's prediction with the user's subjective confidence. However, this comes at the risk of, in the worst case, a consistent underestimation of skill mastery, resulting in redundant training due to the user's low confidence. For this reason, it may be reasonable to explore effective ways to *share the BKT's estimate of skill mastery with the user during training* (e.g. a bar graph representing the skill mastery in [0,100]% range in XR). This way, users can align their self-assessment with BKT's prediction.

To lower users' frustration from excessive practices, there are important factors to consider. Recall that BKT parameters (refer Sec. 8.1) for each skill are tuned *with the assumption that its pre-requisite skills are already mastered.* Hence, if the system carelessly transition the user to a new skill that requires the current one to avoid frustration, this violates the BKT's assumption and, therefore, degrades its prediction accuracy. Furthermore, this transition would also likely overload the user to simultaneously learn the pre-requisite and new skills, potentially incurring more frustration. To circumvent these issues, scaffolding [137] a skill could help users master each skill before transitioning to the next skill, while lowering frustration. This means to use domain knowledge to divide the associated task distribution to the skill into different sections of according to difficulty, and sample from relatively easier section to assist learning. However, this scaffolding may be labor intensive in trade-off.

Anticipated Challenges in Authoring Our System

There are a number of anticipated challenges as external researchers implement or adapt our system for their purpose.

Learning Scenic Probabilistic Programming Language To author our system using Scenic, the developer needs to be familiar with Python programming language because Scenic language is embedded in Python. This means that Scenic supports syntax akin to Python such as for/while-loop, if-else statements, defining functions with `def`, etc. Building on to Python, Scenic adds syntax that resembles natural English such as `ahead of, can see, visible to, behavior`, etc. Hence, we expect developers with background in Python and English be able to easily learn Scenic to model distributions of interactive tasks, although this needs to be investigated through a separate study.

Interfacing Scenic to XR Simulator Scenic is simulator-independent, meaning it is not specifically designed for a particular simulator. Hence, it can be flexibly interfaced to other simulators. The generic code template and the instructions for interfacing is shared in the Scenic documentation [62]. Using the template, Scenic has been interfaced to nearly ten different simulators, including our interface Scenic to Unity in this study.

Limitations of Scenic Language Currently, Scenic does not support modeling human intent. This limits modeling sophisticated *coordinated* interactions between multiple agents. For example, in Echo Arena VR esports we use for our study, it is difficult to model

2 vs 2 offense situation, where a user needs to collaborate with its virtual teammate in order to score against the two opponent players. The two opponent defenders and the teammate need to adapt their strategies, or coordinated behaviors, to the user's strategy, which is unknown a priori and needs to be inferred in real-time based on observations of the user's behavior. Simply assigning a distribution over behaviors over, for instance, the virtual teammate *without inferring intent* may frequently result in generating unrealistic coordination. Yet, Scenic does not provide relevant syntax and semantics to model intent. Consequently, in our study, the training tasks involve limited interaction with at most one virtual player. One suggestion is to extend Scenic syntax to express gestures, which are commonly used to explicitly communicate intent.

8.3.8 Limitations of the Study

There are a number of limitations in our study.

Too much variability degrades learning: Although introducing variability in training tasks has been shown to induce better learning and generalization of psychomotor skills [195, 43, 171, 15], too much variability in training can actually impair learning [23]. In our study, we assume that the instructors who design the task distributions would introduce adequate amount of variations to train each skill. We do not have any mechanism in place to measure and determine whether the size of variations in the provided task distribution would negatively impact learning.

Extracting Tacit Knowledge: It can be challenging to extract *tacit* domain knowledge from the experts to specify accurate evaluation metrics per task distribution. We do not experience this issue in our study, but we foresee this may be an issue depending on the skill to train. We have not investigated a methodological approach to cope with this difficult problem.

Tangled Effects in Our Study: And, in our study, the effects of personalizing the curriculum and the training speed are *jointly* taking place. To better evaluate the isolated effect of the two independent variables, further ablation study is necessary.

8.4 Bibliographic Notes

Our work relates to three fields of study: adaptive training, intelligent tutoring systems, and, more broadly, psychomotor skill training. Pertaining to adaptive training, previous literature [176, 175, 174] explore adjusting the training environments, such as varying a basketball hoop's heights and sizes, to generate incrementally difficult training exercises. Some have explored the *gamification* of training with pre-defined difficulty levels in VR [178, 71]. Others also investigated the personalization of a curriculum to increase the effectiveness of training [126, 157]. However, these literature do not address how to systematically and adaptively predict skill mastery with respect to a distribution of training tasks. Instead, they employ ad hoc measures to predict mastery such as solving N fixed number of consecutive training tasks, or allowing users to adaptively self-assess their skill after completing each task. In our study, we compare BKT to self-assessment as it is, to the best of our knowledge, currently the only *adaptive and scalable* methodology to predict skill mastery with respect to task distributions.

The mechanism for skill mastery prediction is better addressed in the field of intelligent tutoring systems (ITS) [167] from education research. ITS personalizes training contents, such as curricula and training speeds, to each student's needs. Also, it makes use of Bayesian knowledge tracing (BKT) [198] to predict each student's skill mastery. However, ITS and BKT are traditionally designed to train purely cognitive skills like algebra, without involving any motor skills. Nevertheless, because psychomotor skills consist of cognitive skills with motor executions, our work is inspired by ITS. The design of our training system is built on top of curriculum personalization framework introduced in [186, 121]. These work provide a generic framework to assess skills and personalize curriculum without relying on any dataset a priori. We adapt the curriculum generation aspect of this work to incorporate BKT to estimate skill mastery to further adapt the training speeds.

There are numerous research directions in training psychomotor skills. Some focuses on the construction of a high fidelity training simulators [83, 149]. Another investigates diverse forms of feedback to correct the users by exploring visual, tactile, and auditory haptic feedback [193, 168, 36], developing new media like augmented mirror [7], or accounting for social interactions [180] and cognitive science [84]. Others design new physical devices (e.g. airRacket [173], tacTower [112]) to enhance sensory realism and engagement in training. In comparison, our work assumes training in XR where an XR simulator and any feedback mechanisms are already provided. In this setting, we focus on how to personalize, or individually adapt, the training contents, specifically the curriculum and the training speed.

8.5 Chapter Summary

We derive principles from learning sciences and neuro-physiology to develop a personalized training algorithm for h-CPS in XR. In particular, we train psychomotor skills to humans to better control systems and efficiently achieve tasks. A structured variations in training tasks have been shown to promote generalization of skills in neuro-physiology. We use Scenic to model and generate these variations of tasks in XR. To personalize the training, we derive algorithms from intelligent tutoring systems in learning sciences to model human knowledge. In particular, we model knowledge as a knowledge graph and uses BKT to determine whether a user mastered each node, i.e. skill, in the graph. Given a set of skills to train and a corresponding set of Scenic programs, it adaptively sequences the programs to personalize the curriculum, and adjusts the number of tasks to sample per program to personalize training speed. The experiment results show that our algorithm induces much more consistent learning compared to self-guided learning.

Chapter 9

Final Words

In this thesis, we make a case that a domain-specific PPL, Scenic, is an adequate formalism to model and generate increasingly complex operating scenarios of autonomous systems. Indeed, we demonstrate the generality of this formalism across different domains. The crux of this thesis is to address the challenges that are introduced by adopting Scenic as an environment formalism.

In the context of simulation-based testing and analysis, we devise a machinery for system-level testing that integrates multi-objective falsification with parallelized simulation to efficiently explore distributions of scenarios modeled as probabilistic programs. Then, we zoom into component-level analysis to synthesize probabilistic programs that intuitively explains in which distributions of scenarios the DNN-based component fails.

For sim-to-real validation, Scenic serves as a consistent formalism across simulation and reality, which models scenarios where a component or a system fails. With this formalism, we first propose a formal scenario-based approach to select test scenarios for track testing to robustly induce system failures in reality. To overcome the labor-intensive and non-scalable shortcomings of track testing, we propose a data-driven algorithm to query labelled sensor data with probabilistic programs.

Finally, we demonstrate that we can train autonomous systems in a targeted manner, guided by their failures identified in simulation. In particular, we propose offline and online training techniques for reinforcement learning agents using Scenic. Also, we introduce personalized training algorithm to test and train psychomotor skills to humans in XR, which is another media for simulation.

9.1 Future Work

This thesis is a prelude to exciting future research directions.

9.1.1 Understanding Training Data and Generating Missing Data

Nowadays, sensor data, such as RGB camera, LiDAR, or radar, are essential to develop autonomous systems. As a result, massive amounts of sensor data are being collected, labelled, and even open-sourced. Yet, previous literature have repeatedly shown that not only quantity of sensor data but also the diversity of their contents greatly affect generalization [184, 136]. For example, suppose training data for a self-driving car does not contain any contents of data pertaining to unprotected left turn scenario at an intersection. Then, this may be potentially a hazardous scenario for the self-driving car to safely interact with other traffic participants in. However, given the terabytes of data being collected and labelled, how can we efficiently understand what contents (of reactive and dynamic scenarios) are currently represented in the training data? Having humans to look through these data one by one certainly is not a scalable and cost-effective approach to understand the contents of sensor data, although this regrettably is the status quo.

The sensor data query algorithm we introduce in Ch. 6 provides a data-driven approach to help developers or researchers better understand the contents of sensor data at scale. Leveraging their domain knowledge, researchers can query the labelled sensor data to understand how much of certain contents, or scenarios, are represented. This query with scenarios, modelled as probabilistic programs, helps to abstract terabytes of sensor data to intuitive, abstract scenarios which promotes better understanding of the data. To realize this practical benefit, first, our proposed sensor data query algorithm needs to be extended to query dynamic scenarios with high efficiency. Then, this would open exciting research directions toward human-computer interaction (HCI), specifically human-in-the-loop program synthesis, where HCI algorithms collaborate with humans to efficiently abstract big sensor data to understandable scenarios.

This enhanced understanding of data at scale can contribute to more robust autonomy. By identifying which scenarios of data are not present in the data, we can guide data collection in reality. Or, a more cost-effective approach may be to encode those missing scenarios as probabilistic programs to generate those contents in simulation. These generated synthetic data can be *adapted* to be more realistic through domain adaptation techniques [184] to enhance performance on real data.

9.1.2 Verifying Human Robot Interactions in XR

It is difficult to separate autonomy from interactions with humans. However, the current standard development process for autonomous systems tend to initially overlook this process. When developing from scratch, researchers and developers leverage simulation to design and analyze these systems, where human behaviors are modelled and simulated. Once the systems demonstrate a sufficient level of performance in simulation, the standard next step is to physically implement the system, which incurs much time, labor, and cost. *What is missing in between this step is to validate whether assumed human models, which the robots are designed for, are indeed correct.* If the human models turn out to be not valid in reality, then subsequent modification of already implemented systems may likely incur substantial cost and labor. In this context, XR provides an important "intermediate" step or setting to validate human models. In particular, *in XR, we no longer need models of human behaviors* since actual humans will directly interact with robots. In short, XR relaxes the assumptions of humans made during simulation-based system design process and helps examine their validity. Being able to detect any mis-assumptions and iterate system repair early on, prior to physical implementation, will likely lower the cost of developing robotics in general. Guided by these identified mis-assumptions, formal methods can be used to design systems to more safely interact with humans [146, 179]. Hence, the use of XR to detect any errors in the models in early stages of development would be hugely beneficial. Note that domain-specific PPLs can serve as consistent formalism to model and generate the same scenarios across simulation to XR. *Therefore, the techniques and algorithms proposed in this thesis could be leveraged to test and train both autonomous systems and humans to efficiently and safely interact with each other.*

9.2 Bibliographic Notes

We would like to acknowledge and thank people, organizations, and funding sources that influenced us to shape our future directions. Our future direction on understanding the contents of big sensor data is motivated by our collaborations with American Automobile Association (AAA) and LG Electronics. We thank our collaborators, Daniel Fremont and Yash Pant, from UC Berkeley, Atul Acharya, Xantha Bruso, and Paul Wells from AAA, and Steve Lemke, Qiang Lu, and Shalin Mehta from LG Electronics. We thank Eunsu Ryu, a former principal engineer at Cruise, who shaped our thinking to understand the importance of the contents of sensor data. We would like to also thank Prof. James Weimer from Vanderbilt University to build upon our work in XR to effectively design and verify human robot interactions in XR. Lastly, we are deeply grateful for the National Science Foundation (NSF) graduate fellowship, NSF VehiCaL, Defense Advanced Research Projects Agency (DARPA) Assured Autonomy, which provided us the freedom to pursue our passion.

Bibliography

[1] 4Active Systems. *4Active Surfboard Platform*. 2020. URL: http://www.4activesystems.at/en/products/test-equipment/4activesb.html (cit. on p. 49).

[2] Houssam Abbas, Georgios Fainekos, Sriram Sankaranarayanan, Franjo Ivančić, and Aarti Gupta. “Probabilistic Temporal Logic Falsification of Cyber-Physical Systems”. In: *ACM Trans. Embed. Comput. Syst.* 12.2s (May 2013). ISSN: 1539-9087. DOI: 10.1145/2465787.2465797 (cit. on p. 27).

[3] Houssam Abbas, Matthew O’Kelly, Alena Rodionova, and Rahul Mangharam. “Safe at any speed: A simulation-based test harness for autonomous vehicles”. In: *International Workshop on Design, Modeling, and Evaluation of Cyber Physical Systems*. Springer. 2017, pp. 94–106 (cit. on p. 55).

[4] Amina Adadi and Mohammed Berrada. “Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)”. In: *IEEE Access* 6 (2018), pp. 52138–52160. DOI: 10.1109/ACCESS.2018.2870052 (cit. on pp. 1, 8).

[5] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Aida Nematzadeh Andrew Brock, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. “Flamingo: A Visual Language Model for Few-Shot Learning”. In: *Neural Information Processing Systems (NeurIPS)*. 2022 (cit. on p. 4).

[6] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 2020 (cit. on p. 1).

[7] Fraser Anderson, Tovi Grossman, Justin Matejka, and George Fitzmaurice. “YouMove: enhancing movement training with an augmented reality mirror”. In: *Computer Human Interaction (CHI) Conference on Human Factors in Computing Systems*. 2013 (cit. on p. 103).

[8] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. “S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Parosh Aziz Abdulla and K. Rustan M. Leino. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 254–257. ISBN: 978-3-642-19835-9 (cit. on p. 27).

[9] Michał Antkiewicz, Maximilian Kahn, Michael Ala, Krzysztof Czarnecki, Paul Wells, Atul Acharya, and Sven Beiker. “Modes of Automated Driving System Scenario Testing: Experience Report and Recommendations”. In: *WCX SAE World Congress Experience*. SAE International, 2020 (cit. on p. 55).

[10] *Apollo: Autonomous Driving Solution*. http://apollo.auto/ (cit. on pp. 27, 49).

[11] Hugo Araujo, Gustavo Carvalho, Mohammad Reza Mousavi, and Augusto Sampaio. “Multi-objective Search for Effective Testing of Cyber-Physical Systems”. In: *Software Engineering and Formal Methods*. Ed. by Peter Csaba Ölveczky and Gwen Salaün. Cham: Springer International Publishing, 2019, pp. 183–202. ISBN: 978-3-030-30446-1 (cit. on p. 27).

[12] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. “Employing Multi-Objective Search to Enhance Reactive Test Case Generation and Prioritization for Testing Industrial Cyber-Physical Systems”. In: *IEEE Transactions on Industrial Informatics* 14.3 (2018), pp. 1055–1066. DOI: 10.1109/TII.2017.2788019 (cit. on p. 27).

[13] Abdus Salam Azad, Edward Kim, Qiancheng Wu, Kimin Lee, Ion Stoica, Pieter Abbeel, and Sanjit A Seshia. “Programmatic Modeling and Generation of Real-time Strategic Soccer Environments for Reinforcement Learning”. In: *The Association for the Advancement of Artificial Intelligence (AAAI)* (2021) (cit. on pp. 8, 75).

[14] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. “Satisfiability Modulo Theories”. In: *Handbook of Satisfiability*. IOS Press, 2021. Chap. 33 (cit. on p. 59).

[15] Daniel A. Barun, Ad Aertsen, Daniel M. Wolpert, and Carsten Mehring. “Motor Task Variation Induces Structural Learning”. In: *Current Biology* 19 (4 2009) (cit. on pp. 86, 102).

[16] S. Beiker. *SAE EDGE Research Report: Unsettled Issues in Determining Appropriate Modeling Fidelity for Automated Driving Systems Simulation*. 2019. URL: https://www.sae.org/publications/technical-papers/content/epr2019007/ (visited on 03/02/2020) (cit. on p. 56).

[17] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. “Pyro: Deep Universal Probabilistic Programming”. In: *J. Mach. Learn. Res.* 20 (2019), 28:1–28:6 (cit. on p. 73).

[18] Ilse R. de Boer, Maxim D. Lagerweij, Paul R. Wesselink, and Johanna M. Vervoorn. “The Effect of Variations in Force Feedback in a Virtual Reality Environment on the Performance and Satisfaction of Dental Students”. In: *Aviation, Space, and Environmental Medicine*. Vol. 76. 2005, pp. 352–356 (cit. on p. 86).

[19] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984 (cit. on p. 34).

[20] Rasmus Bro and Age K. Smilde. “Principal component analysis”. In: *Analytical Methods* (2014) (cit. on p. 16).

[21] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016) (cit. on pp. 79, 148).

[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: *arXiv arXiv:1903.11027* (2019) (cit. on pp. 58, 69).

[23] Marco Cardis, Maura Casadio, and Rajiv Ranganathan. “High variability impairs motor learning regardless of whether it affects task performance”. In: *Journal of Neurophysiology*. 2018 (cit. on p. 102).

[24] Alexandra Carpentier, Alessandro Lazaric, Mohammad Ghavamzadeh, Remi Munos, and Peter Auer. “Upper-Confidence-Bound Algorithms for Active Learning in Multi-armed Bandits”. In: *Algorithmic Learning Theory*. Ed. by Jyrki Kivinen, Csaba Szepesvari, Esko Ukkonen, and Thomas Zeugmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 189–203. ISBN: 978-3-642-24412-4 (cit. on pp. 21, 22).

[25] O. Carsten, N. Merat, V. Janssen, E. Johansson, M. Fowkes, and K. Brookhuis. “Human machine interaction and safety of traffic in Europe”. In: *HASTE Final Report* 3 (2005) (cit. on p. 55).

[26] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. “Activation Atlas”. In: *Distill* (2019). DOI: 10.23915/distill.00015 (cit. on p. 40).

[27] Luis I. Reyes Castro, Pratik Chaudhari, Jana Tumova, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. “Incremental Sampling-based Algorithm for Minimum-violation Motion Planning”. In: *CoRR* abs/1305.1102 (2013). arXiv: 1305.1102. URL: http://arxiv.org/abs/1305.1102 (cit. on p. 27).

[28] Money Watch from CBS. *Are robot waiters the wave of the future? Some restaurants say yes*. 2023. URL: https://www.cbsnews.com/news/robot-waiters-restaurants-future/ (cit. on p. 1).

[29] Andrea Censi, Konstantin Slutsky, Tichakorn Wongpiromsarn, Dmitry S. Yershov, Scott Pendleton, James Guo Ming Fu, and Emilio Frazzoli. “Liability, Ethics, and Culture-Aware Behavior Specification using Rulebooks”. In: *International Conference on Robotics and Automation (ICRA)* (2019) (cit. on pp. 20, 21).

[30] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. “Argoverse: 3D Tracking and Forecasting with Rich Maps”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (cit. on p. 58).

[31] X. Chang, P. Ren, P. Xu, Z. Li, X. Chen, and A. Hauptmann. "A Comprehensive Survey of Scene Graphs: Generation and Application". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 45. 1. 2023 (cit. on p. 4).

[32] Xieyuanli Chen, Benedikt Mersch, Lucas Nunes, Rodrigo Marcuzzi, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. "Automatic Labeling to Generate Training Data for Online LiDAR-Based Moving Object Segmentation". In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 6107–6114. DOI: 10.1109/LRA.2022.3166544 (cit. on pp. 7, 57).

[33] E. Clarke and J. Wing. "Formal Methods: State of the Art and Future Directions". In: *ACM Computing Surveys* 28 (1996) (cit. on pp. 2, 5, 7, 8).

[34] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. "Leveraging procedural generation to benchmark reinforcement learning". In: *International conference on machine learning*. 2020 (cit. on pp. 75, 84).

[35] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. "Quantifying generalization in reinforcement learning". In: *International Conference on Machine Learning*. 2019 (cit. on pp. 75, 84).

[36] Nuno N. Correia, Raul Masu, William Primett, Stephan Jurgens, Jochen Feitsch, and Hugo Placido da Silva. "Designing Interactive Visuals for Dance from Body Maps: Machine Learning and Composite Animation Approach". In: *Designing Interactive Systems (DIS)*. 2022 (cit. on p. 103).

[37] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. "Generative Adversarial Networks: An Overview". In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 53–65. DOI: 10.1109/MSP.2017.2765202 (cit. on p. 71).

[38] M. Csikszentmihalyi and I. S. Csikszentmihalyi. *Optimal experience: Psychological studies of flow in consciousness*. Cambridge University Press, 1988 (cit. on p. 91).

[39] Krzysztof Czarnecki. *WISE Drive: Requirements Analysis Framework for Automated Driving Systems*. https://uwaterloo.ca/waterloo-intelligent-systems-engineering-lab/projects/wise-drive-requirements-analysis-framework-automated-driving. Accessed: 2020-02-27. 2018 (cit. on p. 55).

[40] Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. "Quantifying conformance using the Skorokhod metric". In: *Formal Methods in System Design* 50.2-3 (2017), pp. 168–206. DOI: 10.1007/s10703-016-0261-8. URL: https://doi.org/10.1007/s10703-016-0261-8 (cit. on pp. 44, 54).

[41] Michael Desmond, Evelyn Duesterwald, Kristina Brimijoin, Michelle Brachman, and Qian Pan. "Semi-Automated Data Labeling". In: *Proceedings of Machine Learning Research (PMLR)* (2021) (cit. on pp. 7, 57).

[42] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. *OpenAI Baselines Code Repo*. https://github.com/openai/baselines. 2017 (cit. on p. 147).

[43] A. K. Dhawale, Maurice A. Smith, and B. P. Olveczky. "The Role of Variability in Motor Learning". In: *The Annual Review of Neuroscience* 40 (2017), pp. 479–498 (cit. on pp. 86, 102).

[44] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16 (cit. on pp. 14, 24, 39).

[45] T. Dreossi, S. Ghosh, X. Yue, K Keutzer, A. Sangiovanni-Vincentelli, and S. Seshia. "Counterexample-guided data augmentation". In: *International Joint Conference on Artificial Intelligence* (2018) (cit. on pp. 7, 17, 75).

[46] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. "VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems". In: *31st International Conference on Computer Aided Verification (CAV)*. 2019, pp. 432–442 (cit. on pp. 5, 8, 15, 17).

[47] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1407–1416 (cit. on p. 147).

[48] Georgios E. Fainekos and George J. Pappas. "Robustness of Temporal Logic Specifications". In: *Formal Approaches to Software Testing and Runtime Verification*. Ed. by Klaus Havelund, Manuel Núñez, Grigore Roşu, and Burkhart Wolff. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 178–192. ISBN: 978-3-540-49703-5 (cit. on p. 46).

[49] Nancy Falchikov and David Boud. "Student Self-Assessment in Higher Education: A Meta-Analysis". In: *Review of Educational Research*. Vol. 59. 4. 1989, pp. 395–430 (cit. on p. 99).

[50] Pedro Faria. *figma: Web Client/Wrapper to the 'Figma API'*. 2023. URL: https://www.figma.com/ (cit. on p. 89).

[51] Shuo Feng, Yiheng Feng, Haowei Sun, Shao Bao, Aditi Misra, Yi Zhang, and Henry X. Liu. "Testing Scenario Library Generation for Connected and Automated Vehicles, Part II: Case Studies". In: *CoRR* abs/1905.03428 (2019). arXiv: 1905.03428. URL: http://arxiv.org/abs/1905.03428 (cit. on p. 55).

[52] Shuo Feng, Yiheng Feng, Chunhui Yu, Yi Zhang, and Henry X. Liu. "Testing Scenario Library Generation for Connected and Automated Vehicles, Part I: Methodology". In: *CoRR* abs/1905.03419 (2019). arXiv: 1905.03419. URL: http://arxiv.org/abs/1905.03419 (cit. on p. 55).

[53] FIFA. *Laws of the Game.* https://ussoccer.app.box.com/s/xx3byxqgodqtl1h15865/file/850765570638. Accessed: 2021-10-01. 2021 (cit. on p. 81).

[54] Foretellix. *Measurable Scenario Description Language.* https://www.foretellix.com/open-language/. 2023 (cit. on p. 4).

[55] Peter I. Frazier. "A Tutorial on Bayesian Optimization". In: *IEEE transactions on pattern analysis and machine intelligence* (2018) (cit. on p. 16).

[56] David Freeman, Robert Pisani, and Roger Purves. "Statistics". In: *WW Norton & Company (4th Edition)* (2007) (cit. on p. 96).

[57] Daniel Fremont, Johnathan Chiu, Dragos D. Margineantu, Denis Osipychev, and Sanjit A. Seshia. "Formal Analysis and Redesign of a Neural Network-Based Aircraft Taxiing System with VerifAI". In: *Computer-Aided Verification (CAV)* (2020) (cit. on pp. 7, 17, 75).

[58] Daniel J. Fremont. "Algorithmic Improvisation". In: *University of California, Berkeley, Technical Report No. UCB/EECS-2019-133*. 2019. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-133.html (cit. on p. 5).

[59] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. "Scenic: A Language for Scenario Specification and Scene Generation". In: *Proceedings of the 40th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*. 2019 (cit. on pp. 5, 11).

[60] Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. "Scenic: a language for scenario specification and data generation". In: *Machine Learning* (2022). URL: https://doi.org/10.1007/s10994-021-06120-5 (cit. on pp. 4, 5, 11, 14, 64, 67, 78, 132, 134).

[61] Daniel J. Fremont, Edward Kim, Yash Vardhan Pant, Sanjit A. Seshia, Atul Acharya, Xantha Bruso, Paul Wells, Steve Lemke, Qiang Lu, and Shalin Mehta. "Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World". In: *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*. IEEE, 2020, pp. 1–8. DOI: 10.1109/ITSC45102.2020.9294368. URL: https://doi.org/10.1109/ITSC45102.2020.9294368 (cit. on pp. 6, 8, 44, 71).

[62] Daniel J. Fremont, Eric Vin, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. SCENIC *Documentation on the Interfaced Simulators.* https://scenic-lang.readthedocs.io/en/latest/. In the documentation, refer to the "Supported Simulators" tab underneath "Libraries and Simulators". 2021 (cit. on pp. 79, 101).

[63] Rockstar Games. *Grand Theft Auto V.* Windows PC version. 2015. URL: https://www.rockstargames.com/games/info/V (cit. on pp. 14, 32).

[64] Sicun Gao, Soonho Kong, and Edmund Clarke. "dReal: an SMT solver for nonlinear theories over the reals". In: *International Conference on Automated Deduction(CADE)*. 2013 (cit. on pp. 59, 67).

[65] Toni Giorgino. "Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package". In: *Journal of Statistical Software* 31.7 (2009), pp. 1–24. URL: https://doi.org/10.18637/jss.v031.i07 (cit. on pp. 44, 54).

[66] N. Goodman, V. K. Mansinghka, D. Roy, K. Bonawitz, and J. B. Tenenbaum. "Church: A universal language for generative models". In: *Uncertainty in Artificial Intelligence* (2008), pp. 220–229 (cit. on p. 4).

[67] Divya Gopinath, Hayes Converse, Corina S. Pasareanu, and Ankur Taly. "Property Inference for Neural Networks". In: *International Conference on Automated Software Engineering (ASE)* (2019). DOI: https://doi.org/10.1109/ASE.2019.00079 (cit. on p. 31).

[68] Andrew Gordon, T. Henzinger, A. Nori, and Sriram Rajamani. "Probabilistic Programming". In: *Future of Software Engineering Proceedings*. 2014, pp. 167–181 (cit. on pp. 2, 4).

[69] Michael J. Gordon. "A review of the validity and accuracy of self-assessments in health professions training". In: *Academic Medicine*. Vol. 66. 12. 1991, pp. 762–769 (cit. on p. 99).

[70] 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: *International Journal of Computer Vision* 127.4 (2019), pp. 398–414 (cit. on p. 72).

[71] M. Graafland, J. M. Schraagen, and M. P. Schijven. "Systematic review of serious games for medical education and surgical skills training". In: *British Journal of Surgery*. Vol. 99. 10. 2016, pp. 1322–1330 (cit. on p. 102).

[72] Bradley Gram-Hansen, Yuan Zhou, Tobias Kohn, Hongseok Yang, and Frank Wood. "Discontinuous Hamiltonian Monte Carlo for Probabilistic Programs". In: *CoRR* abs/1804.03523 (2018) (cit. on p. 73).

[73] John K Haas. *A history of the unity game engine*. Last accessed August 10th, 2023. 2014. URL: https://digital.wpi.edu/downloads/2f75r821k (cit. on p. 93).

[74] J. H. Halton. "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals". In: *Numerische Mathematik* 2 (1960), pp. 84–90 (cit. on pp. 15, 24).

[75] Sandra G. Hart and Lowell E. Staveland. "Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research". In: *Human Mental Workload*. Ed. by Peter A. Hancock and Najmedin Meshkati. Vol. 52. Advances in Psychology. North-Holland, 1988, pp. 139–183. DOI: https://doi.org/10.1016/S0166-4115(08)62386-9 (cit. on p. 95).

[76] Matthew Hausknecht and et al. “Half Field Offense: An Environment for Multiagent Learning and Ad Hoc Teamwork”. In: *AAMAS Adaptive Learning Agents (ALA) Workshop*. Singapore, 2016 (cit. on p. 85).

[77] M.G. Helander. *The Handbook of Human-Computer Interaction*. Elsevier Science, 2014 (cit. on p. 8).

[78] Mark Hendrikx and et al. “Procedural content generation for games: A survey”. In: *ACM Transactions on Multimedia Computing, Communications, and Applications*. Vol. 9. 2013 (cit. on p. 84).

[79] Matthew D. Hoffman and Andrew Gelman. “The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo”. In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1593–1623 (cit. on p. 73).

[80] Steven Holtzen, Guy Van den Broeck, and Todd D. Millstein. “Dice: Compiling Discrete Probabilistic Programs for Scalable Inference”. In: *Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)* (2020) (cit. on p. 73).

[81] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. “Adversarial Machine Learning”. In: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. AISec ’11. Chicago, Illinois, USA: Association for Computing Machinery, 2011, pp. 43–58. ISBN: 9781450310031. DOI: 10.1145/2046684.2046692 (cit. on pp. 1, 7).

[82] Discord Inc. *Discord*. 2020. URL: https://discord.com (cit. on p. 93).

[83] Ananya Ipsita, Levi Erickson, Yangzi Dong, Joey Huang, Alexa K Bushinski, Sraven Saradhi, Ana M Villanueva, Kylie A Peppler, Thomas S Redick, and Karthik Ramani. “Towards Modeling of Virtual Reality Welding Simulators to Promote Accessible and Scalable Training”. In: *Computer Human Interaction (CHI) Conference on Human Factors in Computing Systems*. 2022 (cit. on p. 103).

[84] Mads Moller Jensen, Majken K. Rasmussen, and Kaj Gronbaek. “Design Sensitivities for Interactive Sport-Training Games”. In: *Designing Interactive Systems (DIS)*. 2014 (cit. on p. 103).

[85] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath N. Sridhar, Karl Rosaen, and Ram Vasudevan. “Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks?” In: *International Conference on Robotics and Automation (ICRA)* (2017) (cit. on p. 71).

[86] Eric Jones, Travis Oliphant, Pearu Peterson, and et al. *SciPy: Open source scientific tools for Python*. 2001. URL: http://www.scipy.org/ (cit. on p. 94).

[87] Niels Justesen, Ruben Rodriguez Torrado, Philip Bontrager, Ahmed Khalifa, Julian Togelius, and Sebastian Risi. “Illuminating generalization in deep reinforcement learning through procedural level generation”. In: *Neural Information Processing (NeuriPS) Workshop on Deep Reinforcement Learning* (2018) (cit. on p. 84).

[88] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie J. Cai, James Wexler, Fernanda B. Viégas, and Rory Sayres. "Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)." In: *ICML*. Ed. by Jennifer G. Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 2673–2682. URL: http://dblp.uni-trier.de/db/conf/icml/icml2018.html#KimWGCWVS18 (cit. on p. 40).

[89] Edward Kim, Divya Gopinath, Corina Pasareanu, and Sanjit Seshia. "A Programmatic and Semantic Approach to Explaining and Debugging Neural Network Based Object Detectors". In: *Conference on Computer Vision and Pattern Recognition*. IEEE, 2020, pp. 11125–11134. DOI: 10.1109/CVPR42600.2020.01114 (cit. on pp. 8, 17, 28).

[90] Edward Kim, Zachary Pardos, Sanjit Seshia, and Bjoern Hartmann. "A principled intelligent occupational training of psychomotor skills in virtual reality". In: *University of California, Berkeley, Technical Report No. UCB/EECS-2023-17*. 2023. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-17.html (cit. on pp. 8, 86).

[91] Edward Kim, Jay Shenoy, Sebastian Junges, Daniel J Fremont, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. "Querying Labelled Data with Scenario Programs for Sim-to-Real Validation". In: *International Conference on Cyber Physical Systems (ICCPS)* (2022) (cit. on p. 8).

[92] S. Kirkpatrick, JR. C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* (1983) (cit. on p. 16).

[93] LLC Kisling Nestico & Redick. *Self-driving Car Accident Statistics*. 2022. URL: https://www.knrlegal.com/car-accident-lawyer/self-driving-car-accident-statistics/ (cit. on p. 1).

[94] Ron Koymans. "Specifying real-time properties with metric temporal logic". In: *Real-time systems* 2.4 (1990), pp. 255–299 (cit. on p. 48).

[95] Friedrich Kruber, Jonas Wurst, and Michael Botsch. "An Unsupervised Random Forest Clustering Technique for Automatic Traffic Scenario Categorization". In: *21st International Conference on Intelligent Transportation Systems (ITSC)* (2018) (cit. on p. 55).

[96] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. "Automatic Variational Inference in Stan". In: *Neural Information Processing Systems(NIPS)*. 2015 (cit. on p. 73).

[97] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. "Automatic Differentiation Variational Inference". In: *J. Mach. Learn. Res.* 18 (2017), 14:1–14:45 (cit. on p. 73).

[98] Volodymyr Kuleshov and Doina Precup. "Algorithms for the multi-armed bandit problem". In: *Journal of Machine Learning Research* (2000) (cit. on p. 22).

[99] T. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. K. Mansinghka. “Picture: A probabilistic programming language for scene perception”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 4390–4399 (cit. on p. 4).

[100] T. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. K. Mansinghka. “Quicksand: A lightweight embedding of probabilistic programming for procedural modeling and design”. In: *Neural Information Processing Systems (NeuriPS) Workshop on Probabilistic Programming* (2014) (cit. on p. 4).

[101] Rohit Kundu. *F1 Score in Machine Learning: Intro and Calculation*. 2022. URL: https://www.v7labs.com/blog/f1-score-guide (cit. on p. 34).

[102] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. “Google research football: A novel reinforcement learning environment”. In: *AAAI Conference on Artificial Intelligence*. 2020 (cit. on pp. 80, 81, 147).

[103] Leslie Lamport. “What Good is Temporal Logic?” In: *Information Processing* (1983) (cit. on p. 15).

[104] Virtual Reality Master League. *EchoArena VR Master League*. Statistics on the number of participants can be found by clicking “Extra” tab and then “Statistics”. Last Accessed August 10th, 2023. 2022. URL: https://vrmasterleague.com/EchoArena (cit. on p. 93).

[105] Carol D. Lee. *An Introduction to Vygotsky*. London: Routledge, 2005. Chap. Signifying in the zone of proximal development (cit. on p. 91).

[106] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. “Network randomization: A simple technique for generalization in deep reinforcement learning”. In: *International Conference on Learning Representations*. 2020 (cit. on p. 75).

[107] Kimin Lee, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin. “Context-aware dynamics model for generalization in model-based reinforcement learning”. In: *International Conference on Machine Learning*. 2020 (cit. on pp. 75, 84).

[108] R. Likert. “A technique for the measurement of attitudes”. In: *Archives of Psychology* 22.140 (1932) (cit. on p. 89).

[109] Ming-Yu Liu, Xun Huang, and Jiahui Yu. “Diffusion Models: A Comprehensive Survey of Methods and Applications”. In: *arxiv:2209.00796*. 2023 (cit. on p. 4).

[110] Ming-Yu Liu, Xun Huang, and Jiahui Yu. “Generative Adversarial Networks for Image and Video Synthesis: Algorithms and Applications”. In: *Proceedings of the IEEE*. 2021, pp. 1–24 (cit. on p. 4).

[111] Rosemary Luckin. “Designing children’s software to ensure productive interactivity through collaboration in the zone of proximal development (ZPD)”. In: *Information Technology in Childhood Education Annual* 2001.1 (2001), pp. 57–85 (cit. on p. 91).

[112] Martin Ludvigsen, Maiken Hillerup Fogtmann, and Kaj Gronbaek. "TacTowers: An Interactive Training Equipment for Elite Athletes". In: *Designing Interactive Systems (DIS)*. 2010 (cit. on pp. 86, 103).

[113] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf (cit. on p. 40).

[114] Marlos Machado, Marc Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents". In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 523–562 (cit. on p. 84).

[115] Rupak Majumdar, Aman Mathur, Marcus Pirron, Laura Stegner, and Damien Zufferey. "PARACOSM: A Test Framework for Autonomous Driving Simulations". In: *Fundamental Approaches to Software Engineering* (2019) (cit. on pp. 4, 5).

[116] Oded Maler and Dejan Nickovic. "Monitoring temporal properties of continuous signals". In: *Proc. FORMATS*. 2004 (cit. on p. 21).

[117] Sharad Malik and Lintao Zhang. "Boolean Satisfiability: From Theoretical Hardness to Practical Success". In: *Communications of the ACM*. Vol. 52. 2009, pp. 76–82 (cit. on p. 59).

[118] Moritz Menze and Andreas Geiger. "Object Scene Flow for Autonomous Vehicles". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015 (cit. on p. 58).

[119] B. Milch, B. Marthi, and S. Russell. "Blog: Relational modeling with unknown objects". In: *ICML workshop on statistical relational learning and its connections to other fields* (2004), pp. 67–73 (cit. on p. 4).

[120] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. "Ray: A Distributed Framework for Emerging AI Applications". In: *CoRR* abs/1712.05889 (2017). arXiv: 1712.05889. URL: http://arxiv.org/abs/1712.05889 (cit. on p. 20).

[121] Tong Mu, Shuhan Wang, Erik Andersen, and Emma Brunskill. "Automatic Adaptive Sequencing in a Webgame". In: *International Conference on Intelligent Tutoring Systems (ITS)*. 2021 (cit. on p. 103).

[122] *Multi-application robots usher in the future of how facilities are cleaned*. 2021. URL: https://www.ishn.com/articles/113139-multi-application-robots-usher-in-the-future-of-how-facilities-are-cleaned (cit. on p. 1).

[123] Luis Munoz-Saavedra, Lourdes Miro-Amarante, and Manuel Dominguez-Morales. "Augmented and Virtual Reality Evolution and Future Tendency". In: *Journal of Applied Sciences*. 2020 (cit. on pp. 7, 8).

[124] Zak Murez, Soheil Kolouri, David Kriegman, Ravi Ramamoorthi, and Kyungnam Kim. “Image to Image Translation for Domain Adaptation”. In: *Computer Vision and Pattern Recognition (CVPR)* (2018) (cit. on p. 71).

[125] W. G. Najm, S. Toma, J. Brewer, et al. “Depiction of priority lightvehicle pre-crash scenarios for safety applications based on vehicleto-vehicle communications”. In: *National Highway Traffic Safety Administration (NHTSA)* (2013) (cit. on p. 55).

[126] Laurentiu-Marian Neagu, Eric Rigaud, Vincent Guarnieri, Mihai Dascalu, and Sebastien Travadel. “Selfit v2 – Challenges Encountered in Building a Psychomotor Intelligent Tutoring System”. In: *International Conference on Intelligent Tutoring Systems (ITS)*. 2022 (cit. on p. 102).

[127] NHTSA. *Automated Driving Systems*. https://www.nhtsa.gov/vehicle-manufacturers/automated-driving-systems. Accessed: 2021-09-02. 2021 (cit. on pp. 24, 125, 126).

[128] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. “Gotta learn fast: A new benchmark for generalization in RL”. In: *arXiv:1804.03720* (2018) (cit. on p. 84).

[129] Aditya Nori, Chung-Kil Hur, Sriram Rajamani, and Selva Samuel. “R2: An Efficient MCMC Sampler for Probabilistic Programs”. In: *Association for the Advancement of Artificial Intelligence (AAAI)*. AAAI Press, 2014, pp. 2476–2482 (cit. on p. 73).

[130] Hawkar Oagaz, Breawn Schoun, and Min-Hyung Choi. “Performance Improvement and Skill Transfer in Table Tennis Through Training in Virtual Reality”. In: *IEEE Transactions on Visualization and Computer Graphics*. Vol. 28. 12. 2022 (cit. on p. 86).

[131] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. “Virtual to Real Reinforcement Learning for Autonomous Driving”. In: *Proceedings of the British Machine Vision Conference (BMVC)* (2017) (cit. on p. 71).

[132] Zhongang Qi, Saeed Khorram, and Fuxin Li. “Visualizing Deep Networks by Optimizing with Integrated Gradients”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2019 (cit. on p. 40).

[133] Xin Qin, Nikos Aréchiga, Andrew Best, and Jyotirmoy V. Deshmukh. “Automatic Testing and Falsification with Dynamically Constrained Reinforcement Learning”. In: *CoRR* abs/1910.13645 (2019). arXiv: 1910.13645. URL: http://arxiv.org/abs/1910.13645 (cit. on p. 27).

[134] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106 (cit. on p. 30).

[135] Zahra Ramezani, Johan Lidén Eddeland, Koen Claessen, Martin Fabian, and Knut Åkesson. “Multiple objective functions for falsification of cyber-physical systems”. In: *IFAC-PapersOnLine* 53.4 (2020), pp. 417–422 (cit. on p. 27).

[136] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. "Do ImageNet Classifiers Generalize to ImageNet?" In: *International Conference on Machine Learning (ICML)* (2019) (cit. on p. 105).

[137] K. Ann Renninger and Alexandra List. "Scaffolding for Learning". In: *Encyclopedia of the Sciences of Learning* (2012), pp. 2922–2926 (cit. on p. 101).

[138] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. *Anchor Method Repository*. https://github.com/marcotcr/anchor (cit. on p. 34).

[139] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "Anchors: High-Precision Model-Agnostic Explanations". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*. 2018, pp. 1527–1535. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16982 (cit. on p. 30).

[140] Daniel Richardson. "Some Undecidable Problems Involving Elementary Functions of a Real Variable". In: *J. Symb. Log.* 33.4 (1968), pp. 514–520 (cit. on p. 59).

[141] *Ride with Waymo One*. https://waymo.com/waymo-one/. Last Accessed August 10th, 2023 (cit. on p. 1).

[142] Steven Ritter, John R. Anderson, Kenneth R. Koedinger, and Albert Corbett. "Cognitive Tutor: Applied research in mathematics education". In: *Psychonomic Bulletin and Review*. Vol. 14. 2007, pp. 249–255 (cit. on pp. 87, 90).

[143] C. Roesener, F. Fahrenkrog, A. Uhlig, and L. Eckstein. "A scenario-based assessment approach for automated driving by using time series classification of human-driving behaviour". In: *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)* (2016) (cit. on p. 55).

[144] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. "LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 2020, pp. 1–6. DOI: 10.1109/ITSC45102.2020.9294422 (cit. on pp. 26, 47).

[145] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697 (cit. on p. 90).

[146] Dorsa Sadigh. "Safe and Interactive Autonomy: Control, Learning, and Verification". In: *University of California, Berkeley, Technical Report No. UCB/EECS-2017-143*. 2017. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-143.pdf (cit. on p. 106).

[147] Mikayel Samvelyan and et al. "The StarCraft Multi-Agent Challenge". In: *Workshop on Deep Reinforcement Learning at the 33rd Conference on Neural Information Processing Systems*. 2019 (cit. on p. 85).

[148] Sriram Sankaranarayanan and Georgios Fainekos. "Falsification of Temporal Properties of Hybrid Systems Using the Cross-Entropy Method". In: *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '12. Beijing, China: Association for Computing Machinery, 2012, pp. 125–134. ISBN: 9781450312202. DOI: 10.1145/2185632.2185653. URL: https://doi.org/10.1145/2185632.2185653 (cit. on pp. 16, 24).

[149] M. Schijven and J. Jakimowicz. "Virtual reality surgical laparoscopic simulators". In: *Surgical Endoscopy And Other Interventional Techniques*. 2003 (cit. on p. 103).

[150] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on p. 80).

[151] Scratchapixel. *Ray Tracing: Rendering a Triangle*. https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates. Last Access: 2021-10-23. 2021 (cit. on p. 134).

[152] Hazem Torfah abd Sebastian Junges, Daniel J Fremont, and Sanjit A Seshia. "Formal analysis of AI-based autonomy: from modeling to runtime assurance". In: *International Conference on Runtime Verification* (2021) (cit. on p. 17).

[153] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. 2017, pp. 618–626. DOI: 10.1109/ICCV.2017.74. URL: https://doi.org/10.1109/ICCV.2017.74 (cit. on p. 40).

[154] Younggyo Seo, Kimin Lee, Ignasi Clavera, Thanard Kurutach, Jinwoo Shin, and Pieter Abbeel. "Trajectory-wise Multiple Choice Learning for Dynamics Generalization in Reinforcement Learning". In: *Neural Information Processing Systems* (2020) (cit. on p. 84).

[155] Sanjit Seshia, Ruzena Bajcsy, Thomas Griffiths, Bjoern Hartmann, Shankar Sastry, Richard Murray, Claire Tomlin, and Cynthia Sturton. *VehiCal: Verified Human Interfaces, Control, and Learning for Semi-Autonomous Systems*. https://vehical.org/. Last Accessed August 11th, 2023 (cit. on pp. 9, 86).

[156] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. "Toward Verified Artificial Intelligence". In: *Communications of the ACM* 65.7 (2022) (cit. on pp. 1, 3).

[157] Ka-Chun Siu, Bradley J. Best, Jong Wook Kim, Dmitry Oleynikov, and Frank E. Ritter. "Adaptive Virtual Reality Training to Optimize Military Medical Skills Acquisition and Retention". In: *Military Medicine*. Vol. 181. 2016, pp. 214–220 (cit. on p. 102).

[158] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. "SmoothGrad: removing noise by adding noise". In: *International Conference on Machine Learning (ICML) Workshop on Visualization for Deep Learning* (2017) (cit. on p. 40).

[159] GoMentum Station. https://gomentumstation.net/. Last Accessed August 10th, 2023 (cit. on p. 46).

[160] Douglas. Steinley. "K-means clustering: A half-century synthesis". In: *British Journal of Mathematical and Statistical Psychology* (2010) (cit. on p. 16).

[161] Peter Stone and et al. "Keepaway Soccer: From Machine Learning Testbed to Benchmark". In: *RoboCup 2005: Robot Soccer World Cup IX*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 93–105 (cit. on p. 85).

[162] Kenneth J. Stroud, Deborah L. Harm, and David M. Klaus. "Preflight Virtual Reality Training as a Countermeasure for Space Motion Sickness and Disorientation". In: *Aviation, Space, and Environmental Medicine*. Vol. 76. 2005, pp. 352–356 (cit. on p. 86).

[163] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Sheng Zhao, Shuyang Cheng, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. "Scalability in Perception for Autonomous Driving: Waymo Open Dataset". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (cit. on p. 58).

[164] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic Attribution for Deep Networks". In: *International Conference on Machine Learning (ICML)* (2017). URL: http://arxiv.org/abs/1703.01365 (cit. on p. 40).

[165] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 2018 (cit. on pp. 7, 77).

[166] Daniel Szafir and Bilge Mutlu. "ARTFuL: Adaptive Review Technology for Flipped Learning". In: *Conference on Human Factors in Computing Systems (CHI)*. 2013 (cit. on p. 99).

[167] Albert T.Corbett, Kenneth R.Koedinger, and John R.Anderson. "Intelligent Tutoring Systems". In: *Handbook of Human-Computer Interaction (2nd Edition)*. 1997, pp. 849–874 (cit. on pp. 87, 103).

[168] Richard Tang, Xing-Dong Yang, Scott Bateman, Joaquim Jorge, and Anthony Tang. "Physio@Home: Exploring Visual Guidance and Feedback Techniques for Physiotherapy Exercises". In: *Computer Human Interaction (CHI) Conference on Human Factors in Computing Systems*. 2015 (cit. on p. 103).

[169] Insider Tech. *Robots Could Be Coming To An Airport Near You.* https://www.youtube.com/watch?v=CVqAPK9f3R4. Last Accessed August 10th, 2023. 2017 (cit. on p. 1).

[170] Terry Therneau, Beth Atkinson, and Brian Ripley. *rpart: Recursive Partitioning and Regression Trees.* Version 0.20.2. Feb. 19, 2010. URL: https://github.com/bethatkinson/rpart (cit. on p. 34).

[171] E. Thorp, K. Kording, and F. Mussa-Ivaldi. "Using noise to shape motor learning". In: *Journal of Neurophysiology* (2017) (cit. on pp. 86, 102).

[172] D. Tran, M. D. Hoffman, Rif A. Saurous, Eugene Brevdo, Kevin Murphy, and David M. Blei. "Deep Probabilistic Programming". In: *International Conference on Learning Representations(ICLR).* 2017 (cit. on p. 73).

[173] Ching-Yi Tsai, I-Lun Tsai, Chao-Jung Lai, Derrek Chow, Lauren Wei, Lung-Pan Cheng, and Mike Y. Chen. "AirRacket: Perceptual Design of Ungrounded, Directional Force Feedback to Improve Virtual Racket Sports Experiences". In: *Computer Human Interaction (CHI) Conference on Human Factors in Computing Systems*. 2022 (cit. on p. 103).

[174] Dishita Turakhia, Yini Qi, Lotta-Gili Blumberg, Andrew Wong, and Stefanie Mueller. "Can Physical Tools that Adapt their Shape based on a Learner's Performance Help in Motor Skill Training?" In: *Designing Interactive Systems (DIS).* 2021 (cit. on pp. 86, 102).

[175] Dishita Turakhia, Andrew Wong, Yini Qi, Lotta-Gili Blumberg, Yoonji Kim, and Stefanie Mueller. "Adapt2Learn: A Toolkit for Configuring the Learning Algorithm for Adaptive Physical Tools for Motor-Skill Learning". In: *Designing Interactive Systems (DIS).* 2021 (cit. on pp. 86, 102).

[176] Dishita Turakhia, Andrew Wong, Yini Qi, Lotta-Gili Blumberg, Yoonji Kim, and Stefanie Mueller. "Designing Adaptive Tools for Motor Skill Training". In: *User Interface Software and Technology (UIST) Adjunct*. 2021 (cit. on pp. 86, 102).

[177] Alberto Uriarte and Santiago Ontanon. "A Benchmark for StarCraft Intelligent Agents". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 11. 2. 2021, pp. 22–28. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/12810 (cit. on p. 85).

[178] Marijke Vandermasesen, Tom De Weyer, Peter Feys, Kris Luyten, and Karin Coninx. "Integrating Serious Games and Tangible Objects for Functional Handgrip Training: A User Study of Handly in Persons with Multiple Sclerosis". In: *Designing Interactive Systems (DIS).* 2016 (cit. on p. 102).

[179] Marcell Vazquez-Chanlatte. "Specifications from Demonstrations: Learning, Teaching, and Control". In: *University of California, Berkeley, Technical Report No. UCB/EECS-2022-107.* 2022. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-107.pdf (cit. on p. 106).

[180] Laia Turmo Vidal, Elena Marquez Segura, and Annika Waern. “Movement Correction in Instructed Fitness Training: Design Recommendations and Opportunities”. In: *Designing Interactive Systems (DIS)*. 2018 (cit. on p. 103).

[181] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. *StarCraft II: A New Challenge for Reinforcement Learning*. 2017. arXiv: 1708.04782 (cit. on p. 85).

[182] Kesav Viswanadha, Francis Indaheng, Justin Wong, Edward Kim, Ellen Kalvan, Yash Pant, Daniel J Fremont, and Sanjit A Seshia. “Addressing the IEEE AV Test Challenge with Scenic and VerifAI”. In: *IEEE International Conference on Artificial Intelligence Testing* (cit. on p. 27).

[183] Kesav Viswanadha, Edward Kim, Francis Indaheng, Daniel J. Fremont, and Sanjit A. Seshia. “Parallel and Multi-Objective Falsification with Scenic and VerifAI”. In: *International Conference on Runtime Verification* (2021) (cit. on pp. 8, 17, 19).

[184] Mei Wang and Weihong Deng. “Deep visual domain adaptation: A survey”. In: *Neurocomputing* 312 (2018), pp. 135–153. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.05.083 (cit. on pp. 43, 71, 105).

[185] Peng Wang, Xinyu Huang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. “The apolloscape open dataset for autonomous driving and its application”. In: *IEEE transactions on pattern analysis and machine intelligence* (2019) (cit. on p. 58).

[186] Shuhan Wang, Fang He, and Erik Andersen. “A Unified Framework for Knowledge Assessment and Progression Analysis and Design”. In: *Computer Human Interactions Conference on Human Factors in Computing Systems (CHI)*. 2017 (cit. on p. 103).

[187] Benjamin Weyers, Judy Bowen, Alan Dix, and Philippe Palanque. *The Handbook of Formal Methods in Human-Computer Interaction*. Springer, 2017 (cit. on p. 8).

[188] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. “Feature-Guided Black-Box Safety Testing of Deep Neural Networks”. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2018, pp. 408–426. DOI: 10.1007/978-3-319-89960-2_22 (cit. on p. 40).

[189] David Wingate and Theophane Weber. “Automated Variational Inference in Probabilistic Programming”. In: *CoRR* abs/1301.1299 (2013). arXiv: 1301.1299. URL: http://arxiv.org/abs/1301.1299 (cit. on p. 73).

[190] Hermann Winner, Karsten Lemmer, Thomas Form, and Jens Mazzega. “PEGASUS—First Steps for the Safe Introduction of Automated Driving”. In: *Road Vehicle Automation 5*. Ed. by Gereon Meyer and Sven Beiker. Cham: Springer International Publishing, 2019, pp. 185–195. ISBN: 978-3-319-94896-6 (cit. on p. 55).

[191] Jeffrey Wishart, Steven Como, Maria Elli, Brendan Russo, Jack Weast, Niraj Altekar, and Emmanuel James. "Driving Safety Performance Assessment Metrics for ADS-Equipped Vehicles". In: Apr. 2020. DOI: 10.4271/2020-01-1206 (cit. on p. 20).

[192] Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. "A New Approach to Probabilistic Programming Inference". In: *AISTATS*. Vol. 33. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, pp. 1024–1032 (cit. on p. 73).

[193] Mikołaj P. Woźniak, Julia Dominiak, Michał Pieprzowski, and et al. "Subtletee: Augmenting Posture Awareness for Beginner Golfers". In: *Computer Human Interaction (CHI) Conference on Human Factors in Computing Systems*. 2020 (cit. on p. 103).

[194] Bichen Wu, Forrest N. Iandola, Peter H. Jin, and Kurt Keutzer. "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving". In: *Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 446–454. DOI: 10.1109/CVPRW.2017.60 (cit. on p. 32).

[195] H. G. Wu, Y. Miyamoto, L. Castro, B Olveczky, and M. Smith. "Temporal structure of motor variability is dynamically regulated and predicts motor learning ability". In: *Nature Neuroscience* 17 (2014), pp. 312–321 (cit. on pp. 86, 102).

[196] Huijuan Xu and Kate Saenko. "Ask, Attend and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering". In: *ECCV (7)*. Vol. 9911. Lecture Notes in Computer Science. Springer, 2016, pp. 451–466 (cit. on p. 72).

[197] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning". In: *Computer Vision and Pattern Recognition*. 2020 (cit. on p. 58).

[198] Michael V. Yudelson, Kenneth R. Koedinger, and Geoffrey J. Gordon. "Individualized Bayesian Knowledge Tracing Models". In: *International Conference on Artificial Intelligence in Education (AIED)*. 2013 (cit. on pp. 87, 103).

[199] Amy Zhang, Yuxin Wu, and Joelle Pineau. "Natural environment benchmarks for reinforcement learning". In: *arXiv preprint arXiv:1811.06032* (2018) (cit. on p. 84).

[200] Sicheng Zhao, Bo Li, Xiangyu Yue, Yang Gu, Pengfei Xu, Runbo Hu, Hua Chai, and Kurt Keutzer. "Multi-source Domain Adaptation for Semantic Segmentation". In: *Conference on Neural Information Processing Systems (NeurIPS)* (2019) (cit. on p. 71).

[201] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. "Interpreting Deep Visual Representations via Network Dissection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017) (cit. on p. 40).

[202] Xin Zhou, Xiaodong Gou, Tingting Huang, and Shunkun Yang. "Review on Testing of Cyber Physical Systems: Methods and Testbeds". In: *IEEE Access* 6 (2018), pp. 52179–52194. DOI: 10.1109/ACCESS.2018.2869834 (cit. on p. 27).

Appendix A

Appendix for Chapter 3

For the experiment in Ch. 3, we selected seven scenarios from the list of pre-crash scenarios described by the National Highway Traffic Safety Administration (NHTSA) [127]. The following table describes the seven scenarios.

Scenario #	Scenario Description	Related NHTSA Pre-Crash Scenario(s) [127]
1	The ego vehicle drives straight at a 4-way intersection and must suddenly stop to avoid collision when an adversary vehicle from an oncoming parallel lane makes an unprotected left turn.	Scenario 30: Left Turn Across Path, Opposite Direction
2	The ego vehicle makes an unprotected left turn at a 4-way intersection and must suddenly stop to avoid collision when an adversary vehicle from an oncoming parallel lane drives straight.	Scenario 30: Left Turn Across Path, Opposite Direction
3	The ego vehicle performs a lane change to bypass a leading vehicle before returning to its original lane.	Scenario 14: Changing Lanes, Same Direction
4	A trailing vehicle performs a lane change to bypass the ego vehicle before returning to its original lane.	Scenario 14: Changing Lanes, Same Direction

Table A.1: Part I of Descriptions of 7 Scenarios Selected from NHTSA Pre-crash scenarios [127] for the Experiments in Ch. 3

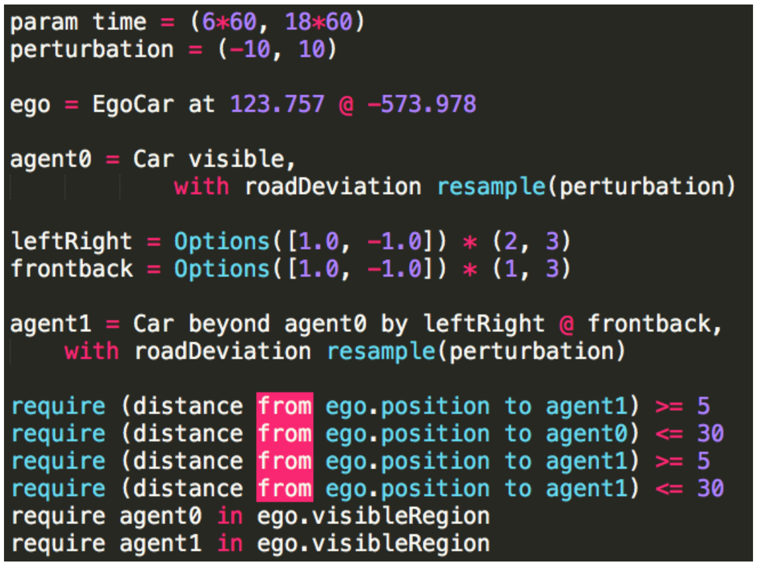
Scenario #	Scenario Description	Related NHTSA Pre-Crash Scenario(s) [127]
5	The ego vehicle performs a lane change to bypass a leading vehicle, but cannot return to its original lane because the leading vehicle accelerates. The ego vehicle must then slow down to avoid collision with the leading vehicle in its new lane.	Scenario 14: Changing Lanes, Same Direction Scenario 20: Rear-End, Striking Maneuver Scenario 22: Rear-End, Lead Vehicle Moving
6	The ego vehicle must suddenly stop to avoid collision when a pedestrian crosses the road unexpectedly.	Scenario 10: Pedestrian, No Maneuver
7	Both the ego vehicle and an adversary vehicle must suddenly stop to avoid collision when a pedestrian crosses the road unexpectedly.	Scenario 10: Pedestrian, No Maneuver

Table A.2: Part II of Descriptions of 7 Scenarios Selected from NHTSA Pre-crash scenarios [127] for the Experiments in Ch. 3

Appendix B

Appendix for Chapter 4

The following are the SCENIC program # 2, 3, and 4 and RGB images rendered using the programs.



```
param time = (6*60, 18*60)
perturbation = (-10, 10)

ego = EgoCar at 123.757 @ -573.978

agent0 = Car visible,
            with roadDeviation resample(perturbation)

leftRight = Options([1.0, -1.0]) * (2, 3)
frontback = Options([1.0, -1.0]) * (1, 3)

agent1 = Car beyond agent0 by leftRight @ frontback,
    with roadDeviation resample(perturbation)

require (distance from ego.position to agent1) >= 5
require (distance from ego.position to agent0) <= 30
require (distance from ego.position to agent1) >= 5
require (distance from ego.position to agent1) <= 30
require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
```

Figure B.1: The SCENIC program for Scenario #2

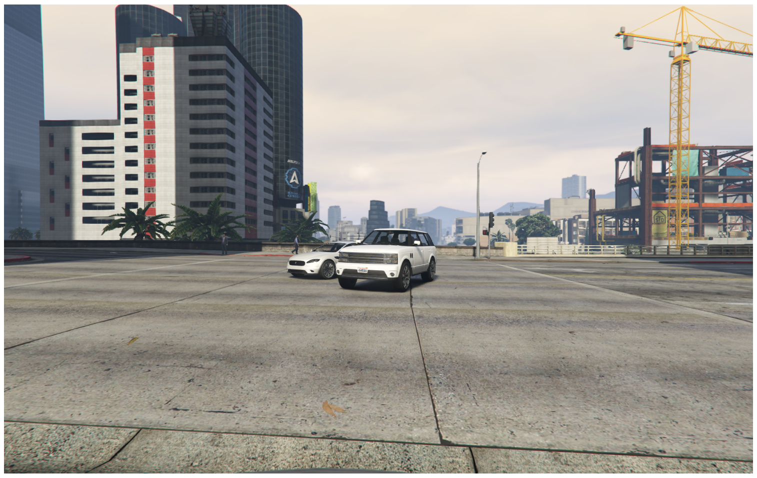
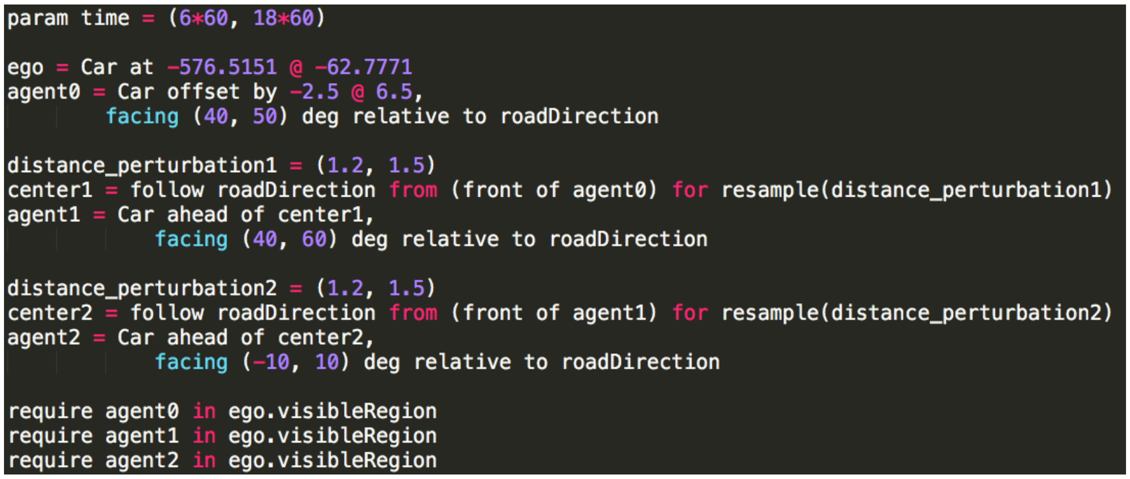


Figure B.2: An RGB image rendered using the program B.1 and GTA-V simulator



```
param time = (6*60, 18*60)

ego = Car at -576.5151 @ -62.7771
agent0 = Car offset by -2.5 @ 6.5,
        facing (40, 50) deg relative to roadDirection

distance_perturbation1 = (1.2, 1.5)
center1 = follow roadDirection from (front of agent0) for resample(distance_perturbation1)
agent1 = Car ahead of center1,
            facing (40, 60) deg relative to roadDirection

distance_perturbation2 = (1.2, 1.5)
center2 = follow roadDirection from (front of agent1) for resample(distance_perturbation2)
agent2 = Car ahead of center2,
            facing (-10, 10) deg relative to roadDirection

require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
require agent2 in ego.visibleRegion
```

Figure B.3: The Scenic program for Scenario #3

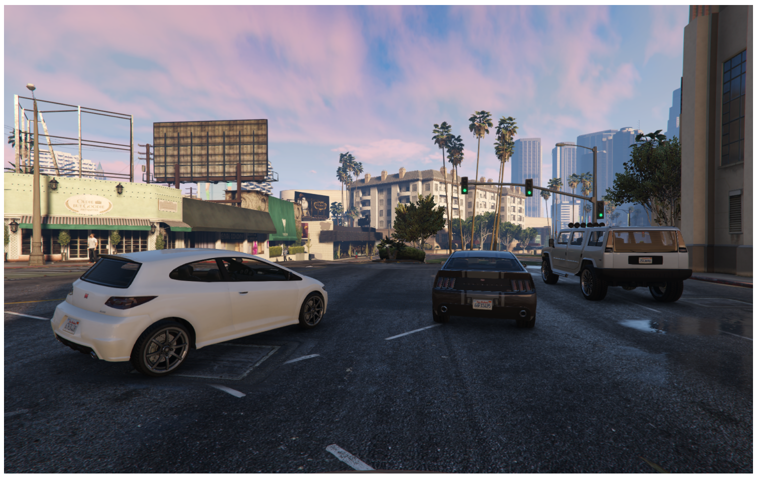
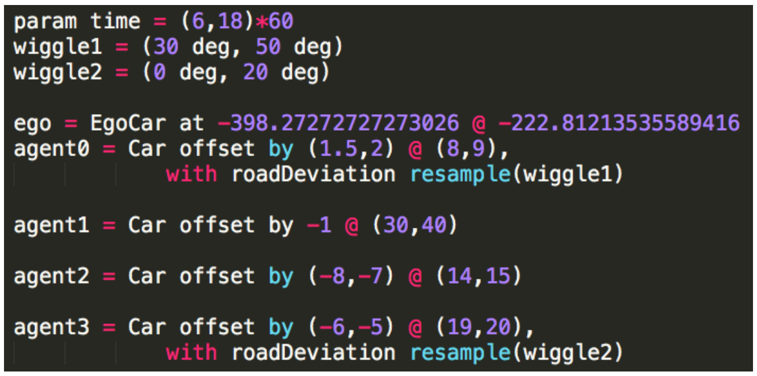


Figure B.4: An RGB image rendered using the program B.3 and GTA-V simulator



```
param time = (6,18)*60
wiggle1 = (30 deg, 50 deg)
wiggle2 = (0 deg, 20 deg)

ego = EgoCar at -398.27272727273026 @ -222.81213535589416
agent0 = Car offset by (1.5,2) @ (8,9),
            with roadDeviation resample(wiggle1)

agent1 = Car offset by -1 @ (30,40)

agent2 = Car offset by (-8,-7) @ (14,15)

agent3 = Car offset by (-6,-5) @ (19,20),
            with roadDeviation resample(wiggle2)
```

Figure B.5: The Scenic program for Scenario #4

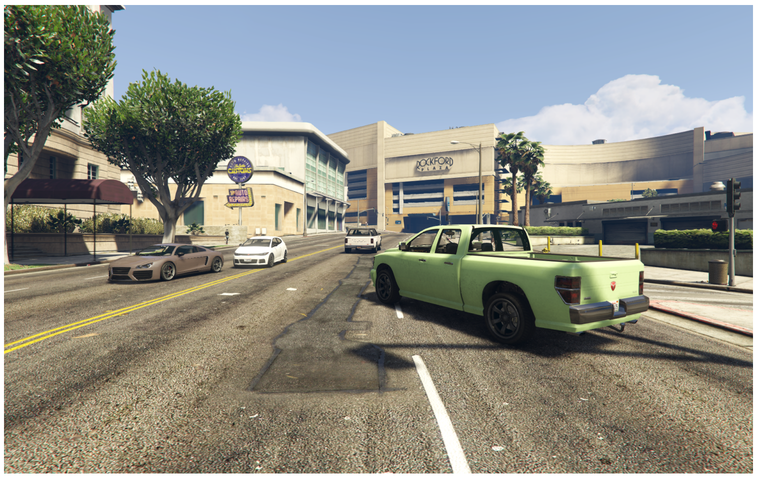


Figure B.6: An RGB image rendered using the program B.5 and GTA-V simulator

Appendix C

Appendix for Chapter 6

We support a large fragment of SCENIC including 26 different position and heading specifiers as well as a wide variety of operators, which together provide an expressive language to flexibly model a diverse range of scenarios. Specifically, the SCENIC fragment supported by our SMT encoding, which enables our query algorithm, includes all (static) SCENIC syntax except for:

1. the operators `following F [from V] for S`,
   `angle [from X] to Y`, and
   `apparent heading of X [from Y]`;
2. `require` statements referring to variables (i.e. semantic features) not present in the label;
3. imports of external Python libraries.

C.1 SMT Encoding

To succinctly present our encoding, we first introduce some notations. First, we denote SCENIC semantics with double brackets, $[\![\cdot]\!]$, and denote the encoding of an expression into an SMT term as $E(\cdot)$. For example, to access a position of a object O as we would in SCENIC, we write $[\![O.position]\!]$ which is equivalent to the object's xy-coordinates, $\langle O_x, O_y \rangle$. SCENIC employs an ego-centric syntax, meaning it requires that `ego` be defined and its syntax, by default, assumes ego as a reference object if not otherwise specified. We will use O to represent a SCENIC object, heading as H, vector (i.e. position) as V. $\top$ denotes True. Basic notations are shown in Fig. C.1.

A SCENIC expression can be categorized into three types: (1) built-in functions, (2) predicates, and (3) distributions. For each expression, we create a new SMT variable, encode any constraints on its value implied by the SCENIC semantics, and gather these variables and encoded constraints throughout each incremental encoding process. For example, to encode the expression `Range(2,5)`, we create a new SMT variable *range1* and add the

$$
\begin{aligned}
\langle x, y \rangle &= \text{point with the given XY coordinates} \\
E(V) &= \langle E(V_x), E(V_y) \rangle \\
E(V_1) \pm E(V_2) &= \langle E(V_{1,x}) \pm E(V_{2,x}), E(V_{1,y}) \pm E(V_{2,y}) \rangle \\
k * (E(V_1) \pm E(V_2)) &= \langle k * (E(V_{1,x}) \pm E(V_{2,x})), k * (E(V_{1,y}) \pm E(V_{2,y})) \rangle, \\
&\quad \text{where } k \in \mathbb{R} \\
\mathit{rotate}\,(\langle x, y \rangle, \theta) &= \langle E(x)\cos(E(\theta)) - E(y)\sin(E(\theta)), \\
&\quad E(x)\sin(E(\theta)) + E(y)\cos(E(\theta)) \rangle \\
\mathit{offsetLocal}\,(O, v) &= [\![\text{O.position}]\!] + \mathit{rotate}\,(v, [\![\text{O.heading}]\!]) \\
\mathit{OP}(V, \theta) &= \texttt{OrientedPoint} \text{ with position V and heading } \theta
\end{aligned}
$$

Figure C.1: Notation used to define the SMT encoding of Scenic syntax.

$$
\begin{aligned}
\mathit{Range}(l, u) &= E(l) \leqslant \mathrm{z} \leqslant E(u) \\
\mathit{Normal}(m, s) &= \top \\
\mathit{Option}(a_1, a_2, ..., a_n) &= (z == E(a_1)) \vee ... \vee (z == E(a_n))
\end{aligned}
$$

Figure C.2: Encoding of Scenic distributions, where z is the SMT variable representing the value sampled from the distribution.

formula $2 \leqslant \mathit{range1} \leqslant 5$ to our set of constraints. The SMT constraints for Scenic's built-in distributions are shown in Fig. C.2.

Second, all the specifiers and operators in our Scenic fragment are built-in functions in Scenic. Therefore, they can be abstractly represented in the following form: $f(a_1, ..., a_k)$ where the function, f, represents the Scenic specifier or operator, and $a_1, ..., a_k$ are input arguments, where each input argument is a Scenic expression. We first encode the input arguments to SMT terms and then encode the function according to the semantics of the specifier or operator as defined in Appendices C.2–C.5 of the Scenic paper [60]. Formally, the encoding is defined by the relation $E(f(a_1, ..., a_k)) = [\![f]\!](E(a_1), ..., E(a_k))$. It is possible that the arguments of f may themselves consist of specifiers and operators with additional input arguments, creating a tree of syntax with its leaf nodes being constants and distributions. In such a case, we traverse down to the leaf nodes and recursively encode the tree toward the root node.

For example, the SMT formula for the position of `otherCar` in the Scenic program in Fig. 6.4 has the form: $[\![\text{ahead of}]\!](E(\mathit{ego.position}), E(\mathit{ego.heading}), E(\mathit{Range}(4, 10)))$. Here, $E(\mathit{ego.position})$ is evaluated as a pair of SMT variables $\langle x, y \rangle$, which must satisfy the constraint $[\![\texttt{On}]\!](\texttt{road}, \langle x, y \rangle)$. The semantics of the `On` predicate are in turn encoded into constraints requiring that $\langle x, y \rangle$ actually lie within the `road` region (as we will see below). Once all the arguments have been encoded as SMT terms (with associated constraints), we

$$\begin{aligned}
\mathit{maxX}(V_1, V_2) &= \mathit{max}(E(V_{1,x}), E(V_{2,x}));\\
&\quad \mathit{maxY}(V_1, V_2) \text{ defined likewise}\\
\mathit{minX}(V_1, V_2) &= \mathit{min}(E(V_{1,x}), E(V_{2,x}));\\
&\quad \mathit{minY}(V_1, V_2) \text{ defined likewise}\\
\mathit{rangeX}(V_1, V_2) &= [\mathit{minX}(V_1, V_2), \mathit{maxX}(V_1, V_2)]\\
\mathit{rangeY}(V_1, V_2) &= [\mathit{minY}(V_1, V_2), \mathit{maxY}(V_1, V_2)]\\
\mathit{slope}(V_1, V_2) &= (E(V_{2,y}) - E(V_{1,y}))/(E(V_{2,x}) - E(V_{1,x}))\\
\mathit{offset}(V_1, V_2) &= E(V_{1,y}) - \mathit{slope}(E(V_1), E(V_2)) * E(V_{1,x})\\
\mathit{lineSeg}(V_1, V_2, x, y) &= \text{point} \langle x, y \rangle \text{ is on the line segment}\\
&\quad (y == \mathit{slope}(V_1, V_2) * x + \mathit{offset}(V_1, V_2)),\\
&\quad \text{if x} \in \mathit{rangeX}(V_1, V_2), \text{y} \in \mathit{rangeY}(V_1, V_2)\\
\mathit{leftLine}(V_1, V_2, x, y) &= \text{point} \langle x, y \rangle \text{ is to the left of the line}\\
&\quad \text{whose direction is } V_1 \text{ to } V_2\\
&= D_x * T_y - D_y * T_x > 0,\\
&\quad \text{where } D = E(V_2) - E(V_1), T = \langle x, y \rangle - E(V_1)\\
\mathit{Disc}(c, r, x, y) &= \text{point} \langle x, y \rangle \text{ is within a disc at } c \text{ of radius } r\\
&= ((x - E(c_x))^2 + (y - E(c_y))^2 \leqslant E(r)^2)\\
\mathit{Sector}(c, r, h, a, x, y) &= \text{point} \langle x, y \rangle \text{is in a sector of } \mathit{Disc}(c, r, \cdot, \cdot)\\
&\quad \text{with heading h and angle a}\\
&= \mathit{Disc}(c, r, x, y) \wedge \mathit{rightLine}(c, V_1, x, y)\\
&\quad \wedge \mathit{leftLine}(c, V_2, x, y),\\
&\quad \text{where } V_1 = \mathit{offsetLocal}(\mathit{OP}(c, h - a/2), \langle 0, r \rangle),\\
&\quad V_2 = \mathit{offsetLocal}(\mathit{OP}(c, h + a/2), \langle 0, r \rangle)
\end{aligned}$$

Figure C.3: Part I of the encoding of region-containment, where $\langle x, y \rangle$ is the point constrained to lie in the region. Disc and Sector regions (which can have random parameters) use the specialized formulas above; all other regions are fixed and use the generic encoding for `On` at the bottom of the figure. This figure continues to Fig. C.4

$$
\begin{aligned}
\mathit{visibleRegion}\,(X,x,y) &= \begin{cases} \mathit{Sector}\,(⟦\text{X.position}⟧, ⟦\text{X.viewDistance}⟧, \\ \qquad ⟦\text{X.heading}⟧, ⟦\text{X.viewAngle}⟧, x, y), \\ \text{if X is } \texttt{OrientedPoint} \\ \mathit{Disc}\,(⟦\text{X.position}⟧, ⟦\text{X.viewDistance}⟧, x, y), \\ \text{if X is } \texttt{Point} \end{cases} \\
\mathit{tri}\,(V_0,V_1,V_2,x,y) &= \text{point}\,\langle x,y\rangle\,\text{is in the triangle defined} \\
&\quad \text{by points } V_0,\ V_1, \text{ and } V_2 \\
&= (\langle x,y\rangle == E(V_1) + (E(V_1) - E(V_2)) * s \\
&\quad + (E(V_2) - E(V_0)) * t), \\
&\quad \text{where } \exists s, \exists t, 0 \leqslant s \leqslant 1, 0 \leqslant t \leqslant 1, s+t \leqslant 1, \\
&\quad \text{using barycentric coordinate system [151]} \\
⟦\mathit{On}⟧(\text{region}, \langle x,y\rangle) &= \bigvee_{i=1}^{n} \mathit{tri}\,(V_0,V_1,V_2,x,y), \\
&\quad \text{for all triangles } (V_0,V_1,V_2) \text{ in a triangulation} \\
&\quad \text{of the region}
\end{aligned}
$$

Figure C.4: Part II of the encoding of region-containment, where $\langle x,y\rangle$ is the point constrained to lie in the region. Disc and Sector regions (which can have random parameters) use the specialized formulas above; all other regions are fixed and use the generic encoding for `On` at the bottom of the figure.

substitute them into ⟦`ahead of`⟧ to obtain the final term for the position of `otherCar`.

Finally, the SMT encoding for the `On` region-containment predicate and associated operations on Scenic regions are shown in Fig. C.3. We encode containment within regions which are fixed (or which become fixed after conditioning) by triangulating the region. For non-fixed regions (discs and sectors), we generate constraints encoding the geometry of the region.

C.2 Proof of Theorem 1

Lemma 1: *Given a fixed object correspondence, the solutions for SMT formula encoding of a* Scenic *program, according to Appendix C.1, are the values of features of the labels that are in the support of the program.*

Proof: The SMT encoding in Appendix C.1 are the encoding of the semantics of Scenic operators, distributions, and regions as stated in [60]. Therefore, given a fixed object correspondence, the requirements specified in a Scenic program and its SMT encoding are equivalent. Hence, the solutions for the variables related to features in the SMT encoding are the values of features of the labels that are in the support of the program.

Proposition 1: *Given a fixed object correspondence, the monolithic encoding algorithm returns `Yes` if and only if the label matches the program with that correspondence.*
Proof: According to the object correspondence, the label provides values for the semantic features of all the objects in the program. Since we disallow `require` statements referring to variables not present in the label, evaluating the requirements with the observed feature values in the label is well-defined. The monolithic encoding translates the Scenic program into a set of requirements, which are mathematically defined as SMT formulas as shown in Appendix C.1. If a requirement is violated, then, by definition, the label does not match the program and, by Lemma 1, the algorithm correctly returns `No`. Otherwise, if all requirements are satisfied, then the label matches the program and, by Lemma 1, the algorithm correctly returns `Yes`.

Proposition 2: *Given a fixed object correspondence, the incremental SMT encoding for a* Scenic *program is equivalent to its monolithic encoding.*
Proof: Suppose there are n semantic features $s_1, \dots, s_n$, in the label and the program. We denote by $\phi_1, \dots, \phi_n$ the corresponding SMT encodings, respectively. For brevity, assume that these SMT formulas include constraints asserting the equality of the observed semantic features to their values in the label. Then the monolithic encoding of the program is equivalent to $\phi_1 \wedge \dots \wedge \phi_n$ (recalling that `require` statements are not included in the SMT encoding).

For simplicity, let's first consider a Scenic program with only dependent semantic features, and no *jointly* dependent features. Suppose our dependency analysis step (refer to Sec. 6.4.2) returns the order $s_1, \dots, s_n$. The dependency order implies a containment relation: for example, if s_2 is dependent on s_1, this means that s_2's expression tree contains that of s_1. Then our incremental encoding is equivalent to $\phi_1 \wedge \phi_2 | \phi_1 \wedge \phi_3 | (\phi_2, \phi_1) \wedge \dots \wedge \phi_n | (\phi_{n-1}, \dots, \phi_1)$ where the vertical bar, $|$, represents conditioning of the semantic features to the corresponding observed values in the label (as discussed in Sec. 6.4.1). The SMT formula $\phi_i | (\phi_{i-1}, \dots, \phi_1)$ results from substituting the expression trees of $s_1, \dots, s_{i-1}$ where they occur in the expression tree of s_i with their corresponding observed values in the label, and then encoding s_i using the resulting tree. Now because ϕ_1 implies that s_1 has its observed value, the substitution above means that $\phi_1 \wedge \phi_2 = \phi_1 \wedge \phi_2 | \phi_1$. Applying this rule iteratively, we obtain $\phi_1 \wedge \dots \wedge \phi_n = \phi_1 \wedge \phi_2 | \phi_1 \wedge \dots \wedge \phi_n | (\phi_{n-1}, \dots, \phi_1)$. Therefore, in this case, the incremental SMT encoding of dependent semantic features is equivalent to the monolithic encoding of the program.

This result extends to the case when semantic features are jointly dependent. Suppose there is a single set of $m \leqslant n$ jointly-dependent semantic features: then for some $i \geqslant 1$ the features $s_i, s_{i+1}, \dots, s_{i+m-1}$ are jointly dependent. Then the incremental SMT encoding is: $\phi_1 \wedge \phi_2 | \phi_1 \wedge \dots \wedge (\phi_i \wedge \phi_{i+1} \wedge \dots \wedge \phi_{i+m-1}) | (\phi_{i-1}, \dots, \phi_1)$

$\wedge\ \phi_{i+m}|(\phi_{i+m-1}, \dots, \phi_1) \wedge \dots \wedge \phi_n|(\phi_{n-1}, \phi_{n-2}, \dots, \phi_1)$

$= \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_{i-1} \wedge (\phi_i \wedge \phi_{i+1} \wedge \dots \wedge \phi_{i+m-1})|(\phi_{i-1}, \dots, \phi_1)$
$\wedge\ \phi_{i+m}|(\phi_{i+m-1}, \dots, \phi_1) \wedge \dots \wedge \phi_n|(\phi_{n-1}, \phi_{n-2}, \dots, \phi_1)$

$= \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_{i+m-1}$
$\wedge\ \phi_{i+m}|(\phi_{i+m-1}, \dots, \phi_1) \wedge \dots \wedge \phi_n|(\phi_{n-1}, \phi_{n-2}, \dots, \phi_1)$

$= \phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_n$
by repeatedly applying the rule $\phi_1 \wedge \phi_2 = \phi_1 \wedge \phi_2|\phi_1$. Finally, this result trivially extends to the case where there is more than one set of jointly-dependent semantic features. Therefore, the incremental SMT encoding is equivalent to the monolithic encoding of the SCENIC program.

Proof of Theorem 1: Consider an iteration of the main loop, in which the object correspondence is fixed. As proven in Proposition 1, for a fixed object correspondence, our algorithm correctly rejects labels which violate any requirements in the SCENIC program. Otherwise, by Proposition 2 the incremental SMT encoding is equivalent to the monolithic one, and so by Proposition 1 the SMT queries will all be satisfiable if and only if the label matches the program for the fixed object correspondence. If so, we return Yes; otherwise we continue the main loop with a new object correspondence. Since we try all possible correspondences (except for those which cannot work because at least one of their incremental SMT formulas will be identical to one which was unsat in an earlier iteration), if there is any correspondence under which the label matches, then the algorithm returns `Yes`. Otherwise, it returns `No`. Hence, the algorithm returns `Yes` if and only if the label matches the program.

C.2.1 SCENIC Programs used for the Efficacy Experiment

The following Fig. C.5 to Fig. C.9 are SCENIC programs of Scenario #1-5 as described in Ch. 6.5.1.

```
ego = Car on drivableRoad,
        facing Range(-15,15) deg relative to roadDirection,
        with visibleDistance 50,
        with viewAngle 135 deg
ped = Pedestrian on roadsOrIntersections,
        with regionContainedIn roadRegion,
        facing Range(-180, 180) deg

require abs(relative heading of ped from ego) > 70 deg
```

Figure C.5: Scenario #1 in the Human Experiment

```
ego = Car on drivableRoad,
        facing Range(-15,15) deg relative to roadDirection,
        with visibleDistance 50,
        with viewAngle 135 deg
other1 = Car on intersection,
            facing Range(50,135) deg relative to ego.heading
other2 = Car on intersection,
            facing -1 * Range(50,135) deg relative to ego.heading

require abs(relative heading of other1 from other2) > 100 deg
require (distance from ego to intersectionRegion) < 10
```

Figure C.6: Scenario #2 in the Human Experiment

```
offset = Uniform(-1,1) * Range(90, 180) deg

ego = Car on drivableRoad,
        facing offset relative to roadDirection,
        with visibleDistance 50,
        with viewAngle 135 deg

otherCar = Car on visible road,
            facing Range(-15, 15) deg relative to roadDirection

require (distance from ego to otherCar) < 10
```

Figure C.7: Scenario #3 in the Human Experiment

```
ego = Car on drivableRoad,
        facing Range(-15, 15) deg relative to roadDirection,
        with visibleDistance 50,
        with viewAngle 135 deg

point1 = OrientedPoint ahead of ego by Range(0,40)
Car at (point1 offset by Range(-1,1) @ 0),
        facing Range(-15, 15) deg relative to roadDirection

oppositeCar = Car offset by (Range(-10, -1), Range(0, 50)),
        facing Range(140, 180) deg relative to ego.heading

point2 = OrientedPoint ahead of oppositeCar by Range(0,40)
Car at (point2 offset by Range(-1,1) @ 0),
        facing Range(-15, 15) deg relative to roadDirection
```

Figure C.8: Scenario #4 in the Human Experiment

```
lanesWithRightLane = filter(lambda i: i._laneToRight, network.laneSections)
egoLane = Uniform(*lanesWithRightLane)

ego = Car on egoLane,
        facing Range(-15,15) deg relative to roadDirection
cutInCar = Car offset by Range(0, 4) @ Range(0, 5),
            facing -1* Range(15, 30) deg relative to roadDirection
```

Figure C.9: Scenario #5 in the Human Experiment

Appendix D

Appendix for Chapter 7

D.1 Description of Proposed Scenarios and Policies

In this section we provide brief descriptions of all of the scenarios in our dataset. To see our Scenic programs, please refer to our attached README pdf file for the pathways to our scenarios.

D.1.1 On Mini and Full Game Scenarios

In general, all our scenarios have the following three termination conditions: (i) ball goes off the field, (ii) change in ball possession across teams, (iii) one of the team scores. If any of these conditions are satisfied, then the scenario will terminate in simulation.

Offense Scenarios

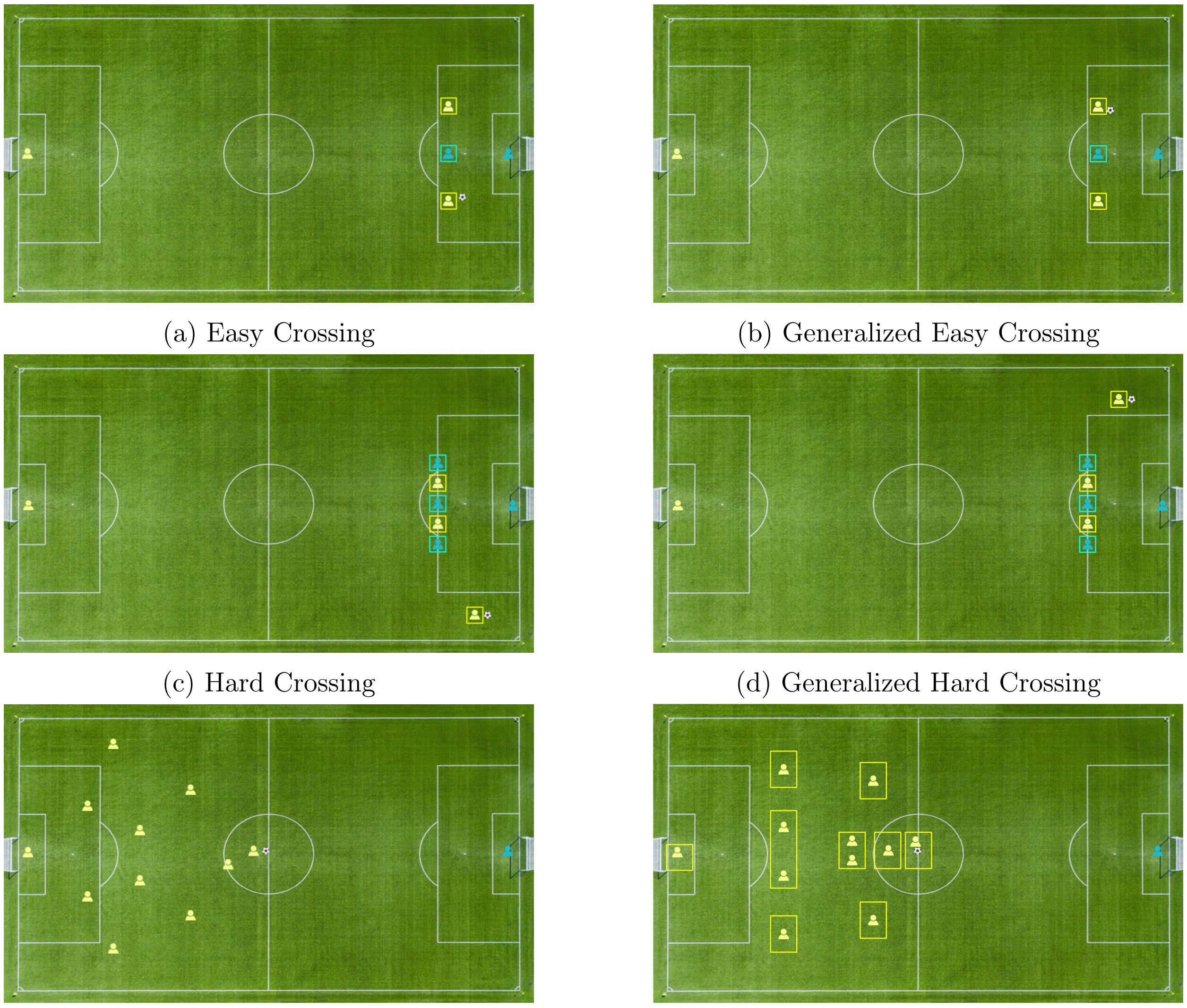
In all of our six offense scenarios as shown in Fig. D.1 and D.2, we explicitly modelled the initial state distribution in using Scenic and implicitly specified the behaviors to environment players by assigning the rule-based AI bots provided by Google Research Football (GRF) to control all non-RL players.

Hard Crossing: A very common scenario in real soccer games: 2 of our players along are guarded by 3 of the opponent players, in an interleaved manner, along the line of the penalty box. Another of our player at the edge of the field is attempting a cross.

11 vs GK: Our team, with a full lineup of eleven players in a traditional 4-4-2 formation, needs to score against the opponent goalkeeper.

Avoid, Pass, and Shoot: Two of our players, one starting on the middle of the right half and the other inside the penalty box, tries to score. One opponent defender starts between our players to intercept direct pass.

Easy Crossing: An easy crossing scenario involving two of our players against opponent defender and goalkeeper in the penalty box.

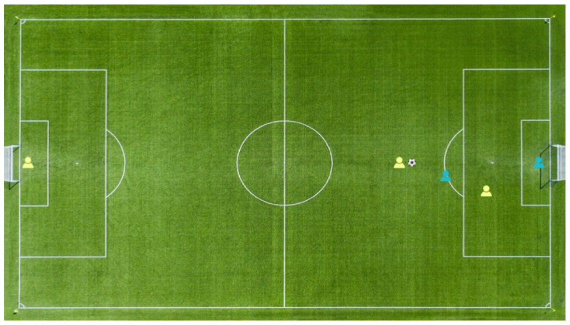


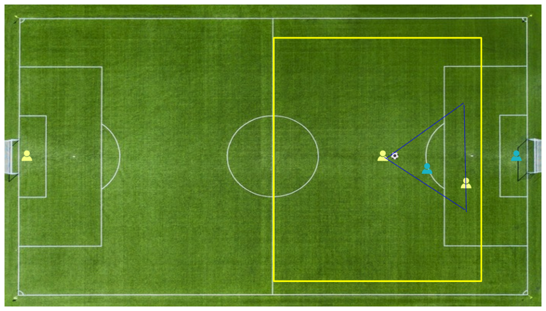
(a) Easy Crossing (b) Generalized Easy Crossing

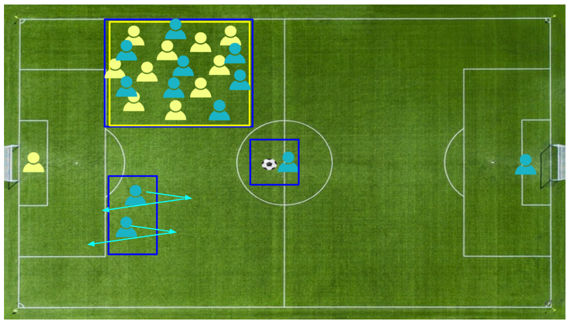
(c) Hard Crossing (d) Generalized Hard Crossing

(e) 11 vs GK (f) Generalized 11 vs GK

Figure D.1: Part I of New offense benchmark scenarios (left images) and corresponding generalized test scenarios (right images) in our dataset. The highlighted boxes represent the regions over which players' initial positions are uniformly randomly distributed. The opponent is in blue and the RL team in yellow.


(a) Avoid, pass, and Shoot


(b) Generalized Avoid, pass, and Shoot


(c) 11 vs 11 with Open Players

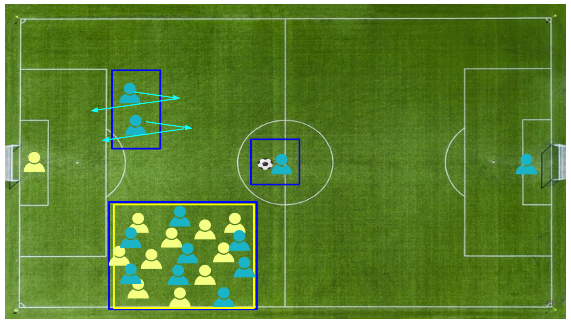

(d) Generalized 11 vs 11 with Open Players

Figure D.2: Part II of New offense benchmark scenarios (left images) and corresponding generalized test scenarios (right images) in our dataset. The highlighted boxes represent the regions over which players' initial positions are uniformly randomly distributed. The opponent is in blue and the RL team in yellow.

11 vs 11 with Open Players: A full game scenario where there are two unmarked players near the opponent goal. This is to test how wide "vision" an RL agent has in identifying unmarked players near the oppnent goal.

Defense Scenarios

Like the offense scenarios, we assigned rule-based AI bots provided by GRF by default to control non-RL players in many of our defense scenarios as shown in Fig. D.3, D.4, D.5. However, if the AI bots do not exhibit expected behavior for our modelled scenarios, we specified non-RL players' behaviors in SCENIC. For these scenarios with specified behaviors in SCENIC, their behaviors are highlighted with light blue arrows.

Goalkeeper vs Opponent: This scenario is designed to train an RL agent to be a defensive goalkeeper when it has to face an opponent one-on-one.

Defender vs Opponent with Hesitant Dribble: The opponent dribbles, stop, then dribbles again in a repeated manner.

Defender vs Opponent with Zigzag Dribble: This opponent aggressively evades the defender with zigzag dribble towards the goal and shoots.

2 vs 2: Typical, 2 vs 2 setting where two defenders are already in place to fend off the two opponents near the penalty area with the ball.

2 vs 2 Counterattack: An opponent attacking midfielder is already advanced deep into the left side of the field. The opponent right midfielder behind either short passes the ball to the attacking midfielder or dribbles up the field.

2 vs 2 High Pass Forward: An opponent attacking midfielder is already advanced deep into the left side of the field. The opponent right midfielder quickly advances the ball to the attacking midfielder via high pass.

3 vs 2 Counterattack: The defender near the penalty box is temporarily outnumbered by the opponent players due to a sudden counterattack.

3 vs 3 Cross from Side: The opponent player on the side crosses the ball to either of the teammates in the middle who are running towards the penalty box to receive the ball. [1]

3 vs 3 Side Build Up Play: Instead of crossing, the opponent player on the side builds up a play by short passing to its teammates.

D.1.2 Testing Generalization

For all the new benchmark scenarios in our dataset as well as for the selected five GRF's scenarios, we generalized those scenarios to test the generalizability of the trained RL agent. Our test scenarios are juxtaposed to corresponding scenarios in Fig. D.1, D.3, D.4, D.5. We modelled these test scenarios by either (i) adding distribution over the initial state or (ii) creating a symmetric opposite formation.

D.1.3 Semi-Expert Stochastic Policies

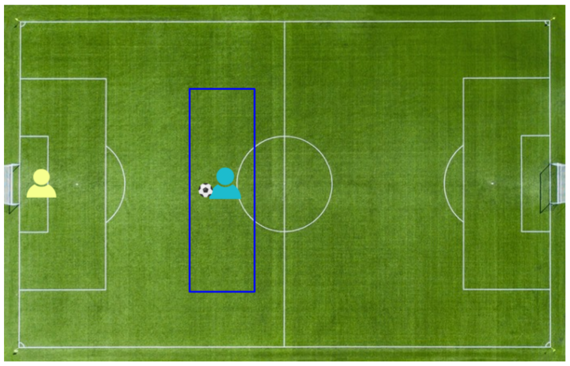
We selected five scenarios from GRF's and our benchmark scenarios. The selected GRF's scenarios are shown in Fig. D.6. The following are the brief descriptions of policies encoded in SCENIC for each scenario.

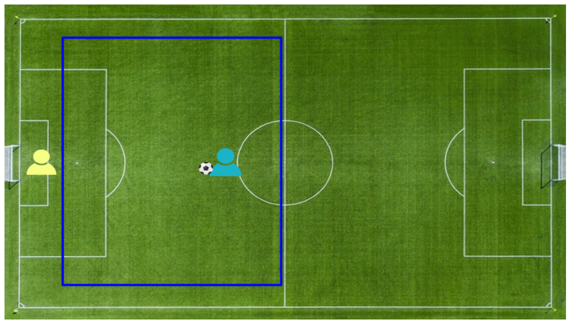
Pass and Shoot with a Goal Keeper: We randomly choose one of the two policies for the RL agent. In the first policy, the player dribbles to the penalty area and shoots once inside it. In the second policy, the player passes the ball to the teammate, who will then dribble towards the goal and shoot.

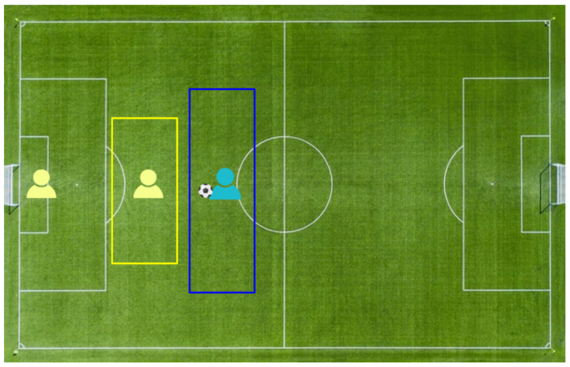
Easy Counterattack: The first player will pass the ball to the right midfielder. Then, the player with the ball will run into the penalty area, and if there is an opponent player on the way, the player will pass the ball to the nearest teammate. The player will shoot at a corner of the goal once inside the penalty area.

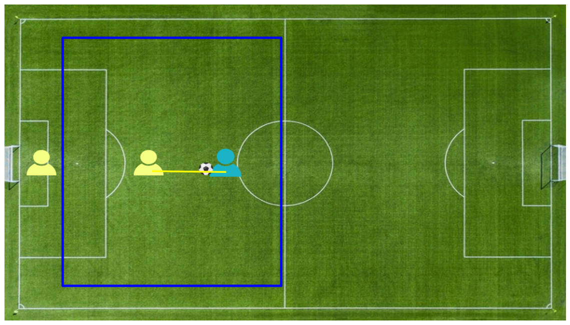
Run to Score with a Goal Keeper: The player with the ball will first sprint towards the goal and turn slightly to either left or right randomly to evade the opponent goalkeeper's

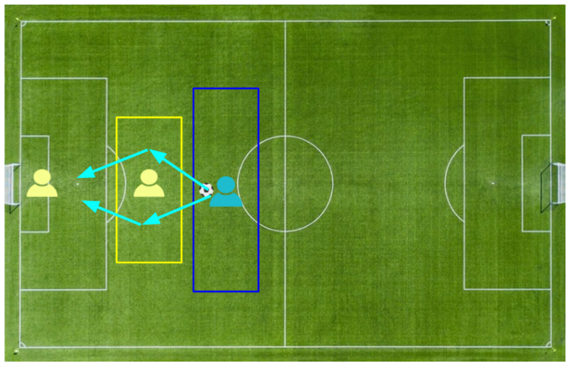
[1] The Defense 3vs3 with cross scenario doesn't conclude a game upon a change in ball possession, unlike other scenarios


(a) Goalkeeper vs Opponent


(b) Generalized Goalkeeper vs Opponent


(c) Defender vs Opponent with Hesitant Dribble


(d) Generalized Defender vs Opponent with Hesitant Dribble


(e) Defender vs Opponent with Zigzag Dribble

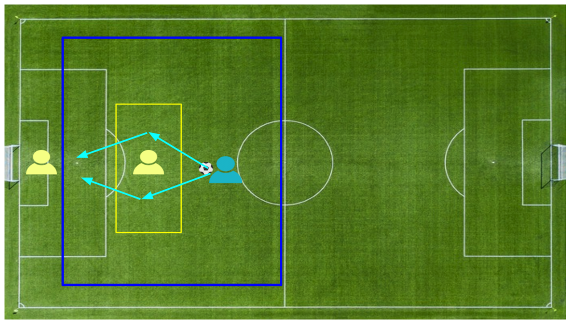
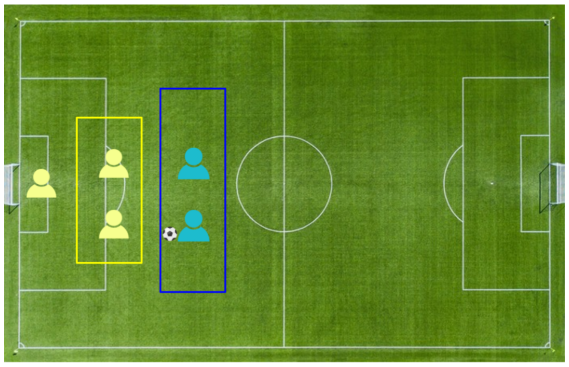
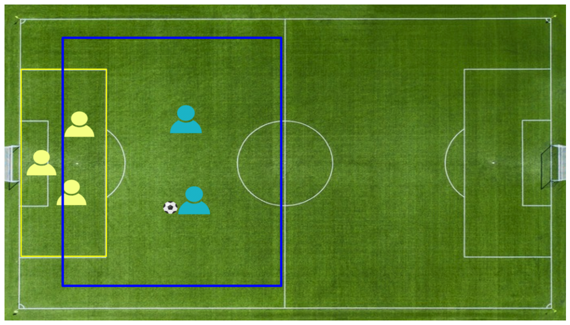
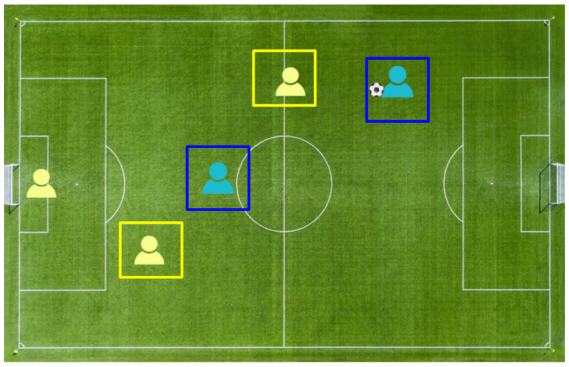
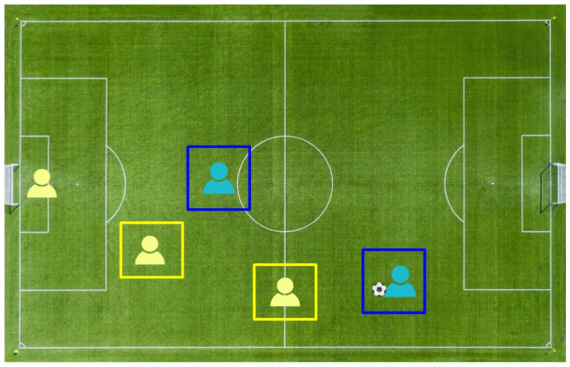

(f) Generalized Defender vs Opponent with Zigzag Dribble

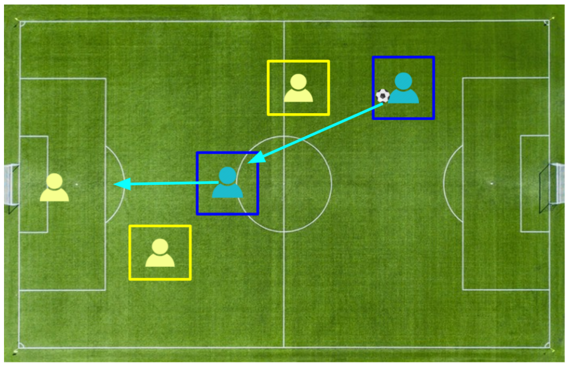
Figure D.3: Part I of New defense benchmark scenarios (left images) and corresponding generalized test scenarios (right images) in our dataset.


(a) 2 vs 2


(b) Generalized 2 vs 2


(c) 2 vs 2 Counterattack


(d) Generalized 2 vs 2 Counterattack


(e) 2 vs 2 with High Pass Forward

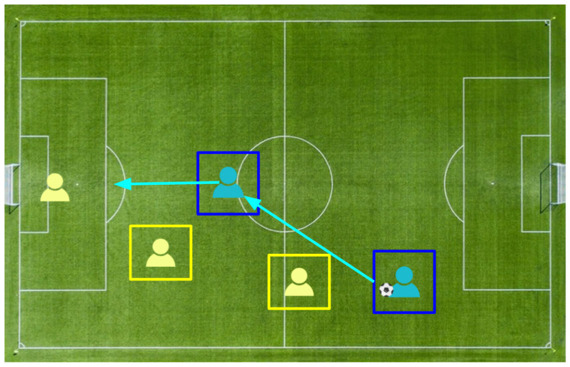
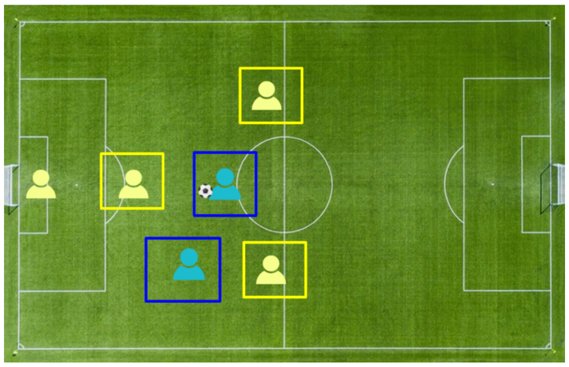
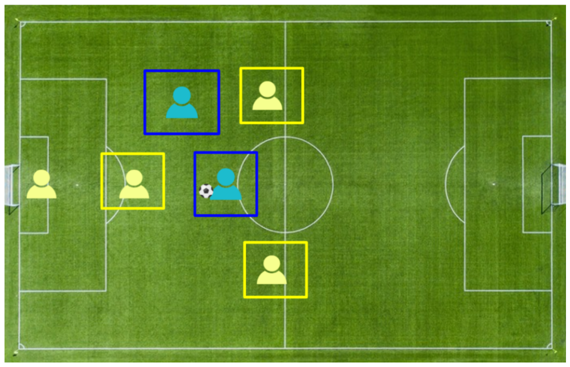
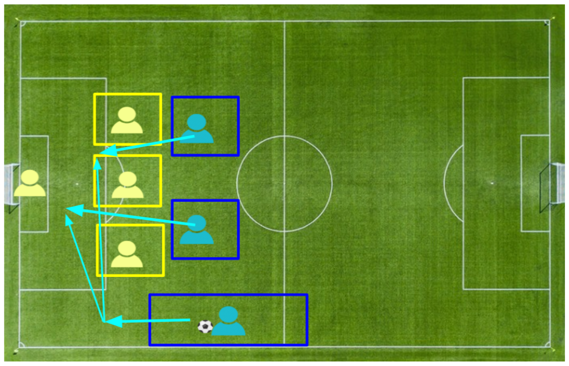
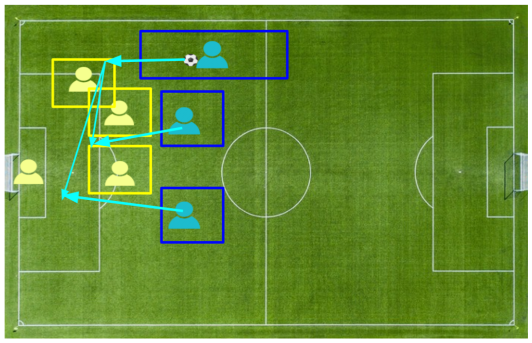

(f) Generalized 2 vs 2 with High Pass Forward

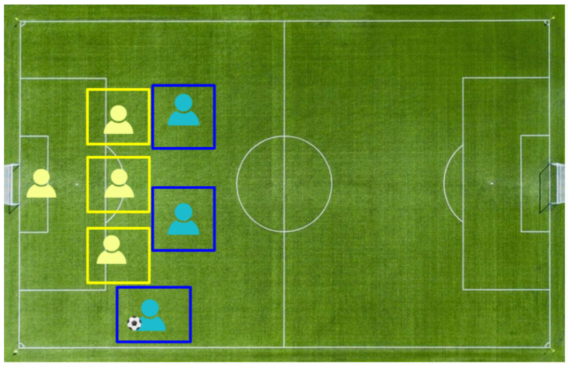
Figure D.4: Part II of New defense benchmark scenarios (left images) and corresponding generalized test scenarios (right images) in our dataset. The highlighted boxes represent the regions over which players' initial positions are uniformly randomly distributed. The opponent is in blue and the RL team in yellow.


(a) 3 vs 2 Counterattack


(b) Generalized 3 vs 2 Counterattack


(c) 3 vs 3 Cross from side


(d) Generalized 3 vs 3 Cross from side


(e) 3 vs 3 side build up play

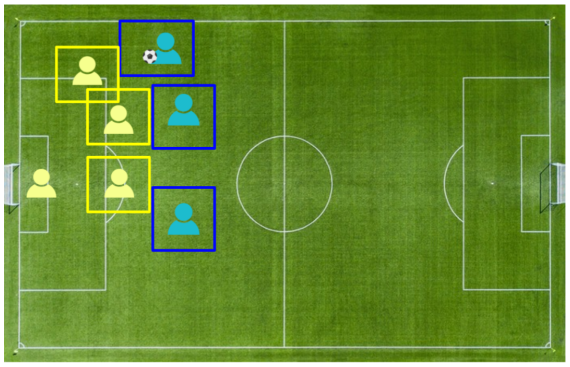
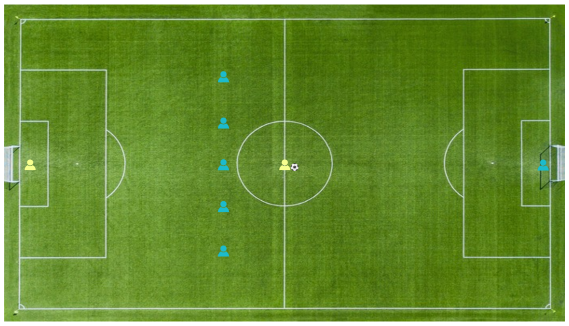
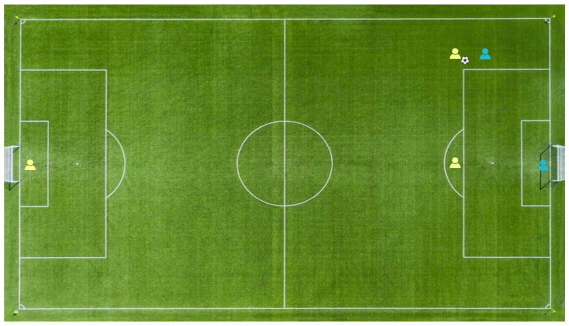

(f) Generalized 3 vs 3 side build up play

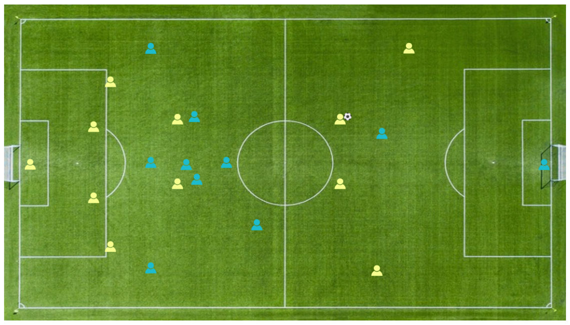
Figure D.5: Part III of New defense benchmark scenario (left images) and corresponding generalized test scenarios (right images) in our dataset. The highlighted boxes represent the regions over which players' initial positions are uniformly randomly distributed. The opponent is in blue and the RL team in yellow.



(a) Run to score with a goal keeper



(b) Pass and Shoot with a Goal Keeper



(c) Easy counterattack

Figure D.6: Google Research Football environment's scenarios for which we wrote semi-expert RL policies

interception. Once the player bypasses the goalkeeper, or is inside the penalty area, the player will shoot.

Avoid Pass and Shoot: The player decides to go towards 3 suitable regions of scoring: left edge/ middle/ right edge of the right goal post, by keeping as much distance possible to the opponent defender. At each time step it decides one of the three destination location. It first predicts its next position for all the three suitable destinations and pick the direction which keeps it farthest of the opponent. If the player comes near the defender it passes the ball to its teammate and if it can successfully go near the goal post, it attempts shooting.

11 vs Goal Keeper: The player with the ball runs towards the right goalpost, if it reaches near the goal post it attempts to shoot. If the opponent goal keeper comes near (i.e. within seven meters) our player before it can reach near the right goal post, it stops running and shoots immediately.

D.2 On Our Scenic Libraries

Users can quickly model scenarios by referencing models, actions, and behaviors from the libraries that we open-sourced along with our interface. To see our library codes, please refer to our attached README pdf file for the pathways to these libraries.

D.2.1 Model Library

The model library defines three categories of objects. First, it defines different regions of the field such as penalty box area. Second, it defines the `Ball`. Lastly, it defines the `Player`. There are two types of player objects which inherit this `class Player`: Left and Right players. The left players represent the RL team, and the right, the opponent. Within each team, the players are further classified into different roles. The naming convention is "the team + role abbreviated in two letters." For example, the left team's goalkeeper is defined as texttttLeftGK. Likewise, for the right team players.

D.2.2 Action Library

This library defines the action space of any players. It consists of twelve different actions such as `Pass`, `Shoot`, `Dribble`, `Sprint`, `Slide`, etc. These actions can be referenced in the Scenic script using the syntax, *take*. For example, to take sliding action, users can write `take Slide()` in their programs.

D.2.3 Behavior Library

The behavior library consists of basic soccer skills that we modelled in Scenic. This library consists of helper functions defined using the syntax, `def`, and behaviors, which reference those helper functions, are defined with the syntax, `behavior`. For brevity, we refer the reviewers to our annotated library code.

D.3 Details on Experimental Setup and Training

We use the OpenAI Baselines' [42] implementation of PPO. The training was run for 5M timesteps with 16 parallel workers. All of our experiments are run on g4dn.4xlarge instances on Amazon AWS: a machine with a single NVIDIA T4 gpu, 16 virtual cores and 64GB RAM.

Network architecture & Hyperparameters For the PPO training, we first experimented with the network architecture and hyperparameters from [102] and was able to reproduce their result. [102] did an extensive search to select their hyperparameters and hence we decided to use the same for our experiments. The architecture we used from [102] is similar to the architecture introduced in [47], with the exception of using four big blocks instead of three.

Table D.1 provides specific values of the hyperparameters used in the PPO experiments.

The parameters for behavior cloning is shown in Table D.2. For the GRF academy scenarios the behavior cloning algorithm is run for 16 epochs while for the offense scenarios it was run for 5 epochs.

Parameter	Value
Action Repetitions	1
Clipping Range	.115
Discount Factor (γ)	0.997
Entropy Coefficient	0.00155
GAE (λ)	0.95
Gradient Norm Clipping	0.76
Learning Rate	0.00011879
Number of Actors	16
Optimizer	Adam
Training Epochs per Update	2
Training Min-batches per Update	4
Unroll Length/n-step	512
Value Function Coefficient	0.5

Table D.1: Training Parameters for PPO.

Parameter	Value
Learning Rate	3e-4
Batch Size	256
Optimizer	Adam
Epsilon(Adam)	1e-5

Table D.2: Training Parameters for Imitation Learning.

D.4 Interface details and Reproducibility

Our interface follows the widely used OpenAI Gym API [21]. For sample usage, we refer readers to our code that is submitted along with this supplement. The code contains necessary scripts, and the attached README pdf file contains detailed description of all our API with examples and a link to a google drive which contains all our trained checkpoints and training logs.

D.5 Performance

As we are adding an additional layer over the GRF simulator, we wanted to measure how much overhead we are adding over the base GRF simulator. We selected five GRF academy scenarios and ran a simulation of 20K timesteps with a random policy both in the GRF simulator and in our interface. The simulation was ran sequentially, i.e., no parallelism was used. Across the scenarios, the GRF simulator took an average of 74.28 seconds for executing a simulation of 20K timesteps, while our interface took 222.07 seconds, showing a 2.99x drop in speed. Some of this overhead is inevitable however, we believe there are ways to speed up . First, as we change the initial state every episode/simulation: we update the

Python scenario file used by the GRF simulator for each episode/simulation. We plan to modify GRF interface to avoid such disk-access each simulation to speed up among other performance improvements. The scenarios we used for the experiments are namely: i) Empty Goal, ii) Empty Goal Close, iii) Pass and Shoot with Keeper, iv) Run, Pass, and Shoot with Keeper, and v) Run to Score with Keeper.

Appendix E

Appendix for Chapter 8

E.1 Participant Background

The participants' backgrounds for the control and the experimental conditions are summarized in Table E.1 and Table E.2, respectively. In both conditions, all participants' ages range in 19 - 25 years old, and none of the participants has experience with Echo Arena esports.

E.2 Details of Our Experiment Design

We record and share video recordings related to each session from a participant's first person view from our study in this hyperlink[1]. The control and the experimental conditions experience the exact same procedure as below, except for the training session as explained in Sec. 8.3.4. Immediately after the pre-test and the training sessions, we ask the participants in both conditions to fill out the NASA TLX survey to measure changes in subjective task load.

Basic Tutorial Session

Because all the participants have never played Echo Arena, we ask them to, first, watch a short tutorial video that we prepared covering basic controls (e.g. thrusts for navigation, grabbing an object, brake). Then, each participant is instructed to wear an Oculus Quest 2 VR headset and familiarize the controls in a few simple training scenarios we created.

Pre-Test Session

We test all ten skills during pre-test. We randomly shuffle the order of skills to lower the chance of learning from sequentially related skills. For each skill, we sample variable number

[1] https://drive.google.com/drive/folders/1r0OzLk0_Ys0rnQpIgZt8MBDqhOlTp_TJ?usp=share_link

ID	Dynamic VR Game Play Frequency	Average Play Time Per Game
C1	less than once a week	1-2 hours
C2	three times a week	2-3 hours
C3	less than once a week	less than 1 hour
C4	less than once a week	1-2 hours
C5	less than once a week	less than 1 hour
C6	less than once a week	less than 1 hour
C7	twice a week	1-2 hour
C8	less than once a week	less than 1 hour
C9	less than once a week	1-2 hours

Table E.1: Control Condition

of evaluation tasks from its corresponding probabilistic program, modeling a distribution of evaluation tasks, and sequentially generate them in VR. This variable number of tasks per skill to sample is set by the experts as explained in Sec. 8.3.4. In this study, coincidentally, this number is three for all the skills. For each task, we display the objective of the task in plain English. After the participant completes each task, we do not provide any feedback as to how they do to avoid learning from the feedback (in contrast, we do provide feedback during training).

Advanced Tutorial Session

Since the training time is limited to 25 minutes, we prepare another tutorial video providing more tips and game intuition to facilitate learning for both conditions. This video provides tips for each skill using training tasks.

Training Session

During training, the control condition, by default, trains with the expert-designed curriculum and they are given the freedom to modify it as they see fit. Per skill, its training tasks are sampled from its probabilistic program, modeling a distribution of training tasks, and generated in VR. The task is displayed in plain English and its time limit for completion is shown on the top right corner of VR display. After completing each task, a feedback is displayed in plain English in VR as to whether the participant successfully solved the task, and, if not, which aspects of the task they fail to solve. Following the feeback, the participants need to respond to our system's binary self-assessment question: "Did you master this skill?" Participants can answer the question by tagging "Yes" or "No" button with a laser from the hand controllers. As they answer this question, a white "Skip" button is displayed below the "Yes" and "No" buttons. If the participant laser tags the "Skip" button, then the participant

ID	Dynamic VR Game Play Frequency	Average Play Time Per Game
E1	less than once a week	less than 1 hour
E2	less than once a week	less than 1 hour
E3	less than once a week	2-3 hours
E4	less than once a week	less than 1 hour
E5	twice a week	1-2 hours
E6	less than once a week	1-2 hour
E7	less than once a week	less than 1 hour
E8	less than once a week	less than 1 hour
E9	twice a week	1-2 hours

Table E.2: Experimental Condition

is immediately transitioned to another skill. Until the participant presses the "Skip" button, regardless of their answer to the binary question, a new task will be sampled and generated for the current skill from its probabilistic program. If participants wish to return to any of the skills they transition from (thereby modifying the default curriculum), they can verbally ask the experiment conductor at any time.

Likewise, for the experimental condition, the training tasks per skill are sampled from their corresponding probabilistic programs, which are the same for both conditions. The task and feedback are displayed in English in VR, just like the control case. Even though unnecessary, the experimental condition is also asked for the same binary self-assessment question as the control for fairness because self-assessment may consume a portion of training time. In contrast to the control, the experimental condition does not have any "Skip" button to transition to another skill. Instead, until BKT predicts mastery for the current skill, training tasks are iteratively sampled from its probabilistic program and sequentially generated in VR. Also, unlike the control, the experimental condition cannot modify the curriculum. Instead, the curriculum and the training speed are all automatically controlled by our training system.

Post Test Session

The post test is exactly the same as the pre-test, except the evaluation tasks for each skill are newly sampled from its corresponding probabilistic program, modeling a distribution of evaluation tasks.

Exit Verbal Interview Session

Our questionnaire is included in the hyperlink provided above.

E.3 Pre-determined Exclusion Criteria for the Experiment

Prior to conducting the study, we determined the following three criteria to exclude participants. Participants who are excluded are financially compensated up to the time they invested in the study at $20 per hour rate. We initially conduct the study with 25 participants, but we end up excluding 7 participants, leaving 18 participants for our study.
Exhibiting Motion Sickness: During any point in the study, if a participant feels any physical discomfort, including commonly known symptoms like nausea or motion sickness, then they are excluded from the study as these symptoms could affect the training or evaluation. We drop 2 participants for this reason (1 from the experimental and 1 from the control condition).
Too Much Skill Expertise: If participants score above 50% on the pre-test (refer to metric in "Learning Gains" of Sec. 8.3.5), they are excluded from the study as their potential learning gains is constricted and, therefore, there is less opportunity to learn new skills. We drop 3 participants for this reason (1 from the experimental and 2 from the control condition).
Extreme Lack of Hand-Eye Coordination: Due to our limited training session of 25 minutes, we exclude participants with extreme lack of hand-eye coordination, who may require much longer time to learn. Prior to pre-test, all the participants equally go through the same tutorial session to familiarize their control. During the pre-test, if participants could not score above 50% on the three most fundamental skills (as shown in Fig. 8.2) that do not have any pre-requisite skill, then we excluded them from the study. The same metric as in "Learning Gains" of Sec. 8.3.5 is used, but only with respect to these three skills, i.e. $K = 3$ instead of 10. These three skills are essential to learn the rest of the skills. We drop 2 participants for this reason (1 from the experimental and 1 from the control condition).

E.4 Skills & Corresponding Training/ Evaluation Task Distributions

In this section, we explain the skills and corresponding training and evaluation task distributions used in our study. We train for ten different skills in our study. Each skill is associated with a training task distribution and an evaluation task distribution. In our experiment, for each skill, the training and the evaluation distributions are equivalent. In other words, the training and the evaluation tasks are sampled from the same probabilistic program per skill. Hence, in our study, each skill is associated with a distinct probabilistic program from which either training or evaluation tasks are sampled.

Although the training and the evaluation task distributions can be different, our recruited experts (refer to Sec. 5.2) suggest that they can be the same for our study since these distributions all consist of wide ranges of continuous distributions. Consequently, the chance of sampling the same task twice is very low, ensuring that the evaluation tasks still examine for generalization of skills to similar yet different tasks from training. The following are brief descriptions of each skill along with its training/evaluation task description. The naming convention for the ten skills we train are shown in the knowledge graph in Fig. 8.2. The video recordings of a participant's first person view of each skill's training/evaluation tasks can be viewed in this hyperlink[2]. To show variations in tasks, for each skill, we share video recordings from pre-test, training, and post-test sessions. All the training/evaluation tasks contain continuous distributions.

Navigation via Thrust (T) Navigate to, brake, and stay within a cyan box whose position is uniformly randomly changing over a pre-designated zone.

Navigation via Grab/Release (GR) Navigate to the goal post only by grabbing and pushing off floating static objects in the zero-gravity space.

Pass to a Stationary Teammate (SP) Accurately pass the frisbee disc to a stationary teammate whose position is uniformly randomly changing over a pre-designated zone.

Navigation via Thrust & Grab/Release (GR_T) Follow the white dotted path. On the path, touch any green floating objects and fly through any yellow hoops. The positions of the green objects and the yellow hoops are uniformly randomly changing while staying within a certain vicinity to each other.

Navigate via Grab/Release to Static Pass (SP_GR) With thrusters disabled, only use grab and release to navigate towards a stationary teammate who is occluded by a large red sphere and then pass the disc. The position of the teammate and the red sphere are uniformly randomly changing over a pre-designated zone. Also, the teammate is always initially positioned to be occluded from the participant.

Pass to a Dynamically Moving Teammate (DP) Accurately pass the disc to a dynamically moving teammate without navigating. The teammate moves in front of the participant. The teammate's initial and destination positions to move from and to are uniformly randomly changing, thereby varying the teammate's directions and distances from the participant.

Navigate via Thrust to Pass to a Moving Teammate (DP_T) The disc is initially placed a uniformly random distance away from the participant who needs to navigate to the disc via thrust and pass it to a moving teammate (whose initial and destination positions are also uniformly randomly varying).

Navigate via Grab/Release to Pass to a Moving Teammate (DP_GR) With thrusters disabled, navigate towards a moving teammate and pass the disc accurately. Similar to DP's training/evaluation tasks, the teammate's initial and destination positions are uniformly randomly varying.

Navigate via Grab/Release with a Sudden Turn to Pass to a Moving Teammate (DP_GR_ST) With thrusters disabled, the participant needs to navigate to a disc which is instantiated uniformly randomly changing distances and directions, quickly turn, and pass to moving player from behind.

Navigate via Thrust and Grab/Release with a Sudden Turn to Pass to a Moving

[2]https://drive.google.com/drive/folders/1senMSlCcIEkZ9qKHP9M3HGUadhkpHgAD?usp=share_link

Teammate (DP_T_GR_ST) Be able to use all the skills learned so far to follow the white dotted path, grab the disc at the end, and pass to a moving teammate before it enters a cyan box. Similar to T_GR, the positions of the green objects and yellow hoops are varying. The distance of the disc from the green object at the end of the white path is varying. Also, the teammate's initial distance to the cyan box is also varying, thereby changing the timing at which it reaches the cyan box.