Scenic4RL: Programmatic Modeling and Generation of Reinforcement Learning Environments

Abdus Salam Azad†, Edward Kim†, Qiancheng Wu, Kimin Lee, Ion Stoica, Pieter Abbeel, Sanjit A. Seshia
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract

The capability of reinforcement learning (RL) agent directly depends on the diversity of learning scenarios the environment generates and how closely it captures real-world situations. However, existing environments/simulators lack the support to systematically model distributions over initial states and transition dynamics. Furthermore, in complex domains such as soccer, the space of possible scenarios is infinite, which makes it impossible for one research group to provide a comprehensive set of scenarios to train, test, and benchmark RL algorithms. To address this issue, for the first time, we adopt an existing formal scenario specification language, SCENIC, to intuitively model and generate interactive scenarios. We interfaced SCENIC to Google Research Soccer environment to create a platform called SCENIC4RL. Using this platform, we provide a dataset consisting of 36 scenario programs encoded in SCENIC and demonstration data generated from a subset of them. We share our experimental results to show the effectiveness of our dataset and the platform to train, test, and benchmark RL algorithms. More importantly, we open-source our platform to enable RL community to collectively contribute to constructing a comprehensive set of scenarios.

1 Introduction

Deep reinforcement learning (RL) has emerged as a powerful method to solve a variety of sequential decision-making problems, including board games [30, 31], video games [23, 35], and robotic manipulation [17]. These successes rely heavily on good simulation environments and benchmarks [2, 3, 5, 6, 33]. However, regardless of a long history of RL benchmarks, the existing RL environments/simulators often fail to generate realistic scenarios portraying a rich variation of environment dynamics. This lack of diversity severely limits the ability of reinforcement learning, primarily resulting in lack of generalization: one of the biggest challenges in deep reinforcement learning [4, 5, 19, 20]. To be able to train robust agents that generalizes over different variations of the environment dynamics, we must train RL agents on scenarios that represent the desired variation.

In literature, several techniques have been adopted to generate a rich variation of learning scenarios [4, 5]; however, none of them provides a support to easily model and generate interactive scenarios. To properly benchmark the capabilities of trained RL agents, we need a systematic way to model and generate scenarios with varying levels of difficulties, which are determined by opponents’ initial conditions and a fined-grained control over their behaviors. Furthermore, collecting quality demonstration data (e.g. to train RL agents or benchmark imitation learning algorithms) becomes expensive because humans have to play the game numerously to generate data, potentially with errors.

†Indicates equal contribution.

Correspondence to Abdus S. Azad (salam_azad@berkeley.edu) and Edward Kim (ek65@berkeley.edu)

Hence, there is a clear need to support users to intuitively model and generate various scenarios at
their demands for accurate benchmark and quality dataset collection.

In this paper, we introduce our platform, Scenic4RL, which offers a systematic and transparent
way of specifying and controlling environment dynamics. We adopt an existing formal scenario
specification language, SCENIC [8, 9], to programmatically model and generate scenarios to bench-
mark RL algorithms. SCENIC provides an intuitive abstraction over environment behaviors to easily
model interactive scenarios. We interfaced SCENIC to Google Research Football [18], an stochastic
environment that requires the agents to learn strategies at multiple levels of abstraction, from learning
to pass, to high level tactics, and presents theoretically infinitely many variations of the environment
dynamics (e.g., player positions and behaviors).

Our contributions are as follows:

- We introduce a platform which supports intuitive modeling and generation of scenarios
  programmatically using SCENIC that enables creating diverse scenarios and easy generation
  of quality demonstration data, at users’ demand.
- We introduce a dataset consisting of i) 13 mini-game scenarios along with reusable model
  and behavior libraries ii) semi-expert stochastic policies (and data generated from them)
  written in SCENIC for select 5 scenarios, and iii) 18 test scripts to evaluate generalization
- We evaluate three use cases of our platform and the dataset by benchmarking scenarios,
  behavior cloning with demonstration data from our stochastic policies, and testing general-
  ization.

2 Background

2.1 Google Research Football Simulator

The Google Research Football (GRF) simulator provides an environment to train and test RL agents
to play soccer. The environment setup is as the following. All the players on the field are controlled
by GRF’s built-in, rule-based AI bots and 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 benchmark RL agent performance.

2.2 Scenario Specification Language: Scenic

SCENIC is an object-oriented, probabilistic programming language which is designed to easily model
and generate scenarios. In SCENIC, a scenario is a distribution over spatio-temporal configuration of
agents and objects. Its natural English-like syntax and semantics let users intuitively specify complex
spatial and temporal relations among objects (see Sec. 3 for an example). Furthermore, SCENIC
allows users to construct object model, action, and behavior libraries and simply reference these
from SCENIC programs. This feature expedites the modeling process because users do not have to
encode a scenario from scratch. We cover the libraries in detail in Sec. 4.1. Finally, users can specify
distributions over states and player behaviors, making a SCENIC program much more expressive.
Prior distribution is specified over an object’s initial states like position and behavior according to
its role. A scenario is generated by sampling an initial state at start and sampling an action for each
object during simulation run time according to their specified behavior. If users instantiates an object
but do not specify any initial states or behavior, those are sampled from the prior distribution on the
object. For further details about the language please refer to [9].

3 The SCENIC4RL Platform for Reinforcement Learning

3.1 The Need for Scenario Specification Language for RL

The capability of an agent depends largely on the quality of the scenarios it is trained on. However,
the existing simulators typically do not provide a systematic way to generate rich-diverse scenarios,
limiting the abilities of reinforcement learning algorithms. For example, the GRF’s scenario python
program, shown in Figure 1(c), used to generate the Pass and Shoot scenario (Fig. 1(a)), initiates
Figure 1: Programs encoding the Google Research Football’s (GRF) pass-and-shoot scenario 

(a) a bird-eye view of the scenario 
(b) a snapshot of GRF environment
(c) GRF’s scenario program

```python
builder.SetBallPosition(0.7, -0.28)
builder.SetTeam(Team.e_Left)
builder.AddPlayer(-1.0, 0.0, e_PlayerRole_GK)
builder.AddPlayer(0.7, -0.3, 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)
```

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

```
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()

LeftMK
attack_midfielder = LeftMK on left_penalty_arc_region,
  facing north,
  with behavior attackMidBehavior()
ego = LeftMK 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)
RightMK
```

Figure 1: Programs encoding the Google Research Football’s (GRF) pass-and-shoot scenario

Each player and the ball at fixed hand-coded positions. There is no support to position the players intuitively by specifying their geometric relations (e.g. “there is a yellow player ahead of a blue player”). Rather the user has to precisely compute initial positions. Also, there are no easy-to-use interface to modify the opponent behaviors. Moreover, the dynamics of the existing environments often lack transparency, which can make reasoning about them difficult in some cases, e.g., it is often not clear how different (or, similar) the train and test environments are [5]. To alleviate these problems, we propose our platform SCENIC4RL, which provides an extensive control over the environment dynamics to generate scenarios on user-demand via SCENIC programs. SCENIC4RL offers a number of advantages over traditional simulators/environments:

1. **Programmable Environments on User-demand:** SCENIC4RL platform allows users to generate scenarios programmatically. Each environment/scenario is represented with a SCENIC program. This enables users to create scenarios as per their necessity.

2. **Incorporation of Domain Knowledge:** SCENIC4RL allows writing stochastic policies which can be used to incorporate domain knowledge and generate offline data.

3. **Enhanced Evaluation:** Users can define reward functions using SCENIC. They can also define arbitrary evaluation metrics to evaluate different aspect of a trained agent, e.g., ball possession, number of intercepts. SCENIC4RL also allows writing simple SCENIC policies to compare with trained RL agents as baselines.

4. **Reusability of libraries:** Users can easily modify or re-use existing SCENIC programs, models, actions, and behaviors to quickly model new scenarios.

5. **Interpretability:** SCENIC provide an intuitive abstract syntax, resembling natural English, which makes the scenario programs interpretable.
6. Transparency: Being able to specify/modify environment dynamics programmatically also improves transparency of the generated scenarios, e.g., users can reason about the difference/similarity of their train and test environments with more clarity.

3.2 Modeling Scenarios with SCENIC

Formally, a scenario is a Markov Decision Process (MDPs) \([32]\) defined as a tuple \((S, A, p, r, \rho_0)\), with \(S\) denoting the state space, \(A\) the action space, \(p(\cdot | s, a)\) the transition dynamics, \(r(\cdot | s, a)\) the reward function, and \(\rho_0\) 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.

Modeling Initial State Distribution Users can intuitively specify initial state distributions with SCENIC’s high-level syntax. For example, refer to the full SCENIC program in Fig. 1(d) which describes a more generalized version of the Pass and Shoot scenario modelled in Fig. 1(c). In line 12-22, the initial state distribution is specified. The SCENIC’s spatial syntax highlighted in yellow lets users intuitively model spatial relations among objects, rather than having to hand-code positions as in the GRF’s scenario. Here, \texttt{Left}\ represents the yellow team, \texttt{Right}\ the blue, and the two following abbreviated capital letters indicate the player role. As discussed in Sec. 2.2, object models (e.g. players, regions, and directions) that are undefined in the program are referenced from the model library, expediting the modeling process. For players instantiated without specification as in line 12 and 22, their state and behavior default to prior distribution, which saves repetitive state specification.

Modeling Transition Dynamics One can flexibly modify transition dynamics of the environment by specifying the behaviors of any players using SCENIC. Take the same example SCENIC program in Fig. 1(d) as above. Line 1-10 models two new behaviors. A behavior can invoke another behavior(s) with syntax \texttt{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 \texttt{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.

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

3.3 Architecture

Figure 2 shows an overview of the overall architecture of SCENIC4RL. The architecture can be divided into two parts, i) RL interface, through which the RL algorithms interact with SCENIC4RL and ii) the SCENIC Server, which executes a SCENIC program and governs the simulation by interacting with the underlying simulator. We implemented our interface using OpenAI Gym API [3], which allows SCENIC4RL 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.
4 Constructing Dataset with the SCENIC4RL Platform

Our dataset comprises of three categories of data: (1) SCENIC libraries, (2) SCENIC programs, and (3) demonstration data generated from semi-expert stochastic SCENIC Policies.

4.1 SCENIC Libraries

As discussed in Sec. 2.2, SCENIC libraries consist of three types: model, action, and behavior libraries. The SCENIC libraries significantly reduces users’ time to model complex scenarios by helping them re-use the set of models, actions, and behaviors in the libraries, rather than having to write 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. Please refer to our open-sourced repository for more details.

(a) 1 vs 1 evasive zigzag towards the goal and shoot
(b) 2 vs 2 advance the ball towards goal via high pass and shoot
(c) 3 vs 3 left mid-fielder crosses to either player in the penalty box
(d) generalization test scenario for pass and shoot scenario in Fig. 1a

Figure 3: Examples of our new defense scenarios with specific assigned behaviors (left three images) and a test scenario to assess generalization (rightmost). 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.

4.2 SCENIC programs

Please refer to our Supplement for detailed descriptions of all our scenarios and policies.
New Benchmark Scenarios We modelled 13 new benchmark scenarios. Nine of them are defense scenarios which are nice complement to GRF’s offense-only scenarios. The remaining four scenarios are additional distinct offense scenarios that enriches GRF’s benchmark scenarios. We varied the initial state distribution and assigned GRF’s AI bot behavior by default. However, if the AI bot did not exhibit the behaviors we expected, we modelled the behaviors of environment agents to create such scenarios. For example, a few of our defense scenarios with specified behaviors are visualized in Fig. 3. In these scenarios, the opponent AI bot did not exhibit expected behaviors so we specified behaviors in the SCENIC programs as described in the sub-captions. In 1vs1 scenario, the opponent AI bot was not aggressive enough to attack the goal and was easily pushed out of the penalty box. In 2vs2 scenario, the opponent initially with the ball did not quickly advance the ball to its teammate closer to the goal post via high pass. In 3vs3, the opponent player with the ball on the leftmost side of the field did not cross the ball to its teammates.

Scenarios for Testing Generalization We provide scripts to test generalization of GRF’s 5 academy scenarios and all of our 13 new benchmark scenarios from our dataset. The scripts test how robust the trained agents are to perturbations in the initial state. We assigned distribution over the initial state while keeping the formation of players and their behaviors in each scenario intact. For example, for testing generalizability of an RL agent trained in the Pass and Shoot scenario (Fig. 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 (Fig. 3d).

Probabilistic Scenic Policies and Offline Dataset SCENIC4RL also allows users to incorporate their domain knowledge by writing probabilistic policies, which can be used as a means to generate rich demonstration data. Users can benefit from using our behavior library to write their policies, instead of having to write them from scratch. In our work, we present simple SCENIC policies for three GRF academy scenarios [18] and two of our new benchmark scenarios in which a proximal policy optimization (PPO) algorithm[28] had relatively poor training performance. These policies are not to be considered “expert”, as they do not solve the scenarios consistently. However, these simple policies can be considered as guidance/semi-expert policies, which can be utilized for better training. We then generate very small datasets of demonstration from these policies, each containing 10K samples collected from the successful (i.e. the agent scores) trajectories. Each sample is a <o, a> pair, i.e., specifying the action a the SCENIC policy took on the observation o. In our experiments we show that, such small offline dataset can expedite the RL training significantly.

4.3 Towards Crowd Sourced Dataset

A current limitation of our benchmark dataset is that it does not provide a comprehensive set of scenarios and policies that cover all realistic soccer scenarios and policies on the field. In fact, there are infinitely many different possible scenarios. We address this issue by open-sourcing our platform and dataset, https://github.com/scenic4RL/scenic4rl/tree/v1, to allow other researchers to contribute their own scenarios and policies as SCENIC programs. A good crowdsourced dataset will cover much more scenarios than a research group can generate. In fact, the effort to crowdsource a dataset in RL domain has been low due to the lack of support for environment modeling. We share our platform to encourage it.

5 Evaluation

In this section, we evaluate our platform SCENIC4RL and the proposed datasets using three RL usecases. First, we benchmark our proposed scenario dataset—using the widely used reinforcement learning algorithm PPO [28]—to enable researchers to compare their algorithms. Then, we evaluate these PPO trained agents using our test dataset, which tests their generalization capabilities against varying initial state distribution. Finally, we evaluate the usefulness of the proposed SCENIC policies by showing significant improvement in training performance.

5.1 Experimental Setup

We run PPO 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 3 different seeds. For all the experiments, we use the stacked Super Mini Map representation for observations
Figure 4: Average Goal Difference of PPO agents on the proposed mini-game scenario Benchmark. The error bars represent 95% bootstrapped confidence intervals.

—a 4x72x96 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 [18]. Similar to the academy scenarios from [18], we also conclude 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 the Supplementary Materials (Section Details on Experimental Setup and Training).

5.2 Mini-game Scenario Benchmark

We benchmark our mini-game scenarios by training agents with PPO for 5M timesteps. Figure 4 shows the average goal differences for all the scenarios. Note that we propose both offense and defense scenarios. For all these 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 a 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. From the graph it can be seen that the proposed scenarios offer a varied levels of difficulties. For example, PPO consistently achieves goal difference of around 0.7 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. Due to the stochasticity of the GRF environment [18], we observed variations in training performance, with the most variation for the 11 VS GK scenario, reaching 0.93 for one run while failing to learn anything for the other two runs.

5.3 Testing for Generalization

As discussed in Sec. 4.2, we introduce a dataset of test scripts to evaluate agents for their robustness against varying initial conditions. Fig. 5 compares the trained agents’ performance in training and test scenarios. As expected, we observe a noticeable drop of performance in the GRF academy and offense scenarios (Fig. 5a). However, for defense scenarios, the drop was not as significant. To further investigate, we evaluated random RL policy performance in the test scenarios as shown in Fig. 5b. Comparing the random and the trained RL agent’s policies in the test scenarios, we observed that the trained RL agent noticeably outperforms the random policy across all test scenarios. This clearly indicates that the trained RL agent policy is generalizing to unseen test scenarios. Note that unlike most of the offense scenarios in Fig. 5a with fixed initial state, all the defense scenarios assigned distributions over the initial state. We conjecture that this difference helped the RL agent to generalize better to unseen yet similar defense scenarios.

5.4 Pretraining with Semi-expert Demonstrations

In these section we evaluate the SCENIC policies and the demonstration data generated by them, as discussed in 4.2. For each scenario, we first use behavior cloning to train an agent using 8K randomly
selected samples from the 10K samples provided in the demonstration dataset. We then continue
the training using PPO for 5M timesteps. Figure 6 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 probabilistic SCENIC policies we
can generate rich quality demonstration data to substantially increase training performance, which
can be particularly useful in practice for environments like GRF which requires a huge amount of
compute resource.

6 Related work

Improving Environment Dynamics for Generalization: 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 [37], introducing
sticky actions [21] have been attempted, but are not robust enough. To ensure generalization, [20]
and [29] 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 [24, 5, 4, 16], typically using techniques such as Procedural Content Generation, 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.

**Formal Scenario Specification Languages** A few scenario specification languages have been proposed in the autonomous driving domain since the introduction of SCENIC. Paracosm language [22] models dynamic scenarios with reactive and synchronous model of computation. The Measurable Scenario Description Language (M-SDL) [7] shares common features as SCENIC to model interactive scenarios. In contrast, however, SCENIC provides a much higher-level, probabilistic, declarative way of modeling, and it is not specific to autonomous driving domain.

**Datasets and Benchmarks.** There is a large body of work focused on designing benchmarks for RL [1–6, 12, 13, 15, 18, 25, 33, 34, 36]. The Arcade Learning Environment (ALE) [2] has becomes a popular benchmark to measure the progress of RL algorithms for discrete control tasks. For continuous control tasks, Duan et al. [6] presented a benchmark with baseline implementations of various RL algorithms. These benchmarks accelerate the progress, resulting advanced RL algorithms [14, 23, 26–28].

Recently, various RL benchmarks designed for various purposes have been proposed. Cobbe et al. [5] presented a suite of game-like environments where the train and test environments differ for evaluating generalization performance of RL agents. Kurach et al. [18] presented a Google Football simulator to study curriculum learning and multi-agent RL. Ray et al. [25] provided a Safety Gym for measuring progress towards RL agents that satisfy the safety constraints. D4RL [12] and RL Unplugged [13] have been proposed to evaluate and compare offline RL algorithms, and Yu et al. [36] proposed Meta-world to study meta- and multi-task RL.

### 7 Conclusion & Future works

In this work we present SCENIC4RL: A platform for generation of diverse scenarios for Reinforcement Learning programmatically, using the SCENIC scenario specification language. We present datasets of mini-game scenarios, behaviors depicting common soccer short-term strategies, probabilistic SCENIC policies and the demonstration dataset generate from them. Lastly, our evaluation demonstrates that our platform and the dataset can be applied to train, test, and benchmark RL algorithms.

**Limitations and future directions.** To the best of our knowledge SCENIC4RL is the first attempt to utilize formal scenario specification language to model environment dynamics in the domain of RL. There are a number of interesting future directions that we plan to pursue.

- Interfacing other simulators: Currently, our platform only supports the Google Soccer Environment. In future we plan to include more simulators.
- As we are adding a layer over the base GRF simulator, we also add an overhead. In future we plan to decrease this overhead. Please refer to the Supplementary Materials for detailed experiment on SCENIC4RL performance.
- Multi-Agent Learning: While our platform does not restrict us to only single-agent RL settings, our current training experiments/benchmark do not include multi-agent training. In future, we would like to introduce scenarios and benchmarks for multi-agent setting.
- Multi-Task Learning: Currently we only train RL agents on a single scenario. However, many soccer skills are transferable. Hence, we want to experiment how we can solve multiple scenarios simultaneously in a multi-task setting
- Use of Formal Analysis Techniques. Using SCENIC4RL it is possible to implement verification and systematic testing techniques (e.g., see [10, 11]).

**Negative Social Impact.** Because our work concerns simulated soccer environment and has no relation to humans in the real world, we do not foresee any negative social impact of this work.
References


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Please refer to Section 7
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Section 5 reports all the experimental results including error bars along with necessary details.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please refer to Section 5.1

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Our platform use Scenic [8, 9] and the Google Research Football simulator [18].
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]