

PGTG: Procedurally Generated Grid-Based Traffic Gym

Joshua Meyer, Felix M. Kuntz, Verena Wolf, Jörg Hoffmann, Timo P. Gros

Keywords: Deep Reinforcement Learning, Benchmarks, Evaluation, Procedural Generation, Exploration, Safety, Generalization

Summary

We present *Procedurally Generated Grid-Based Traffic Gym* (PGTG), a Gymnasium-compliant, feature-rich, extensible, and customizable reinforcement learning benchmark. PGTG requires an agent to navigate procedurally generated tracks while having to manage momentum and being exposed to various stochastic obstacles and the unpredictable behavior of traffic participants. By providing fine-grained control over the map generation, traffic rules, obstacles, and the observation and reward functions, PGTG is a versatile benchmark for various reinforcement learning subfields, such as exploration, safety, and generalization. For instance, PGTG allows decoupling safety from exploration by including subgoals with intermediate rewards. In this paper, we detail the benchmark’s modular design philosophy and provide a comprehensive empirical evaluation using state-of-the-art deep reinforcement learning baselines, highlighting PGTG’s complexity and utility for future research. Our code is available at github.com/neuro-mechanistic-modeling/pgtg.

Contribution(s)

1. This paper presents PGTG, a feature-rich, customizable benchmark designed to isolate or jointly analyze challenges of various deep reinforcement learning subfields (Exploration, Safety, and Generalization) within a single unified framework.
Context: Previous benchmarks are simplistic (Baier et al., 2021), do not provide a unified API and extension interface (Chevalier-Boisvert et al., 2023), or are not easily interpretable (Todorov et al., 2012); and often conflate different metrics (Ray et al., 2019).
2. *Exploration:* By leveraging procedural generation, PGTG creates sparse-reward tasks for inter-episode exploration on a fixed map, and intra-episode exploration by generating one episodically. We show that state-of-the-art intrinsic motivation struggles with the former.
Context: Intrinsic motivation, based on novelty approximations, struggles with procedural generation (Raileanu & Rocktäschel, 2020), and PGTG’s traffic serves as stochastic observation providing ‘fake’ novelty that the algorithm has to identify as such.
3. *Safety:* A decoupled reward-penalty signal and dense navigation rewards with navigation cues isolate safety performance by assisting the agent with the navigation task. We demonstrate that current state-of-the-art safety approaches are challenged by this setting.
Context: We adopt separate reward-cost signals (Ray et al., 2019) allowing us to show that CPO (Achiam et al., 2017) and PPO-Lag (Ray et al., 2019) fail to master this task.
4. *Generalization:* Procedural generation provides a straightforward way to evaluate distribution shifts and generalization performance—be it across track dimensions or other map generation parameters. We show how to measure generalization across track parameters and traffic conditions, and that current algorithms leave significant room for improvement.
Context: Cobbe et al. (2020) point out that using procedural generation is particularly effective to achieve and test generalization, while procedural generation can also be used to define curricula (Portelas et al., 2020; Dennis et al., 2020).

PGTG: Procedurally Generated Grid-Based Traffic Gym

Joshua Meyer^{1,2}, Felix M. Kuntz^{1,2}, Verena Wolf^{1,2}, Jörg Hoffmann^{1,2}, Timo P. Gros^{1,2}

{joshua.meyer, felix_maurice.kuntz, verena.wolf, timo.gros}@dfki.de
hoffmann@cs.uni-saarland.de

¹Saarland Informatics Campus, Universität des Saarlandes, Germany

²Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Saarbrücken, Germany

We present *Procedurally Generated Grid-Based Traffic Gym* (PGTG), a Gymnasium-compliant, feature-rich, extensible, and customizable reinforcement learning benchmark. PGTG requires an agent to navigate procedurally generated tracks while having to manage momentum and being exposed to various stochastic obstacles and the unpredictable behavior of traffic participants. By providing fine-grained control over the map generation, traffic rules, obstacles, and the observation and reward functions, PGTG is a versatile benchmark for various reinforcement learning subfields, such as exploration, safety, and generalization. For instance, PGTG allows decoupling safety from exploration by including subgoals with intermediate rewards. In this paper, we detail the benchmark’s modular design philosophy and provide a comprehensive empirical evaluation using state-of-the-art deep reinforcement learning baselines, highlighting PGTG’s complexity and utility for future research. Our code is available at github.com/neuro-mechanistic-modeling/pgtg.

1 Introduction

In recent years, reinforcement learning (RL) has seen a surge in real-world applicability. Driven by neural networks as universal function approximators, RL can be used on increasingly complex tasks. To support this evolution into *deep* RL (DRL), the research community has invented an entire ecosystem of tools and benchmarks to simplify research by easing the training and performance assessments of DRL agents. Most notably, by conforming to the community-established standard interface Gymnasium (Towers et al., 2024), the various libraries enjoy seamless interoperability. Access to common, easy-to-use tools is key to accelerating research and ensuring comparability.

Recent DRL milestones include achieving superhuman performance in hard-exploration Atari games (Bellemare et al., 2013), mastering complex games like Chess and Go (Silver et al., 2018; Zhang & Yu, 2020), and assisting research or even discovering new results on their own (Fawzi et al., 2022; Hubert et al., 2025). Breakthroughs, like mastering MONTEZUMASREVENGE, demonstrate that significant progress can be made when the community works with shared, reproducible benchmarks.

Despite these successes, DRL agents in complex domains face three intertwined challenges. First, exploration remains difficult when rewards are sparse and state spaces are large. Second, in safety-critical applications like autonomous driving, standard DRL approaches often fail to guarantee safety or satisfy hard constraints, motivating the field of safe RL (García & Fernández, 2015; Zhao et al., 2023). Third, high training performance does not imply robust generalization. Agents frequently exploit shortcuts or spurious correlations, overfitting to the training distribution rather than learning transferable behavior (de Haan et al., 2019; Geirhos et al., 2020; Langosco et al., 2022; Delfosse et al., 2024; Suau et al., 2024). In practice, they co-occur: an autonomous driving agent must explore efficiently, drive safely, and generalize to unseen roads. Studying them in isolation on specialized benchmarks makes it difficult to assess trade-offs and interactions under controlled conditions.

One of the most established benchmarks in this space is RACETRACK (Sutton & Barto, 2018; Baier et al., 2021), which abstracts autonomous driving to navigating a static, grid-based track while managing momentum and accounting for action noise. However, its simplicity limits its utility: (i) static tracks encourage overfitting to positions rather than generalizing over track features; (ii) the absence of other traffic participants prevents studying interactions with unpredictable agents; and (iii) noise is applied uniformly, rather than through avoidable features, limiting the scope for safety-relevant decision-making. While more flexible frameworks like MINIGRID (Chevalier-Boisvert et al., 2023) exist, they provide only a loose structure rather than a unified benchmark. Each environment must be implemented from scratch, leading to API inconsistencies and limited comparability.

We present *Procedurally Generated Grid-Based Traffic Gym* (PGTG), a Gymnasium-compatible benchmark environment that addresses these limitations. In PGTG, tracks are procedurally generated and contain traffic with configurable driver profiles, obstacles mimicking diverse road conditions, and a modular reward structure with optional subgoals. Crucially, PGTG’s features can be granularly activated or deactivated, enabling researchers to study exploration, safety, and generalization both in isolation and in combination. This controlled modularity is a key advantage over using separate benchmarks and enables direct measurement of trade-offs (e.g., how safety constraints affect exploration) without confounds arising from differences in environment design. With an easy installation process, a Gymnasium-compliant interface, and human-interpretable visualizations (see fig. 1), we keep the barrier to entry for PGTG low.

The remainder of this paper is organized as follows: We begin by introducing the necessary preliminaries in section 2. PGTG’s features, design goals, and philosophy are detailed in section 3. In section 4, we demonstrate the utility of PGTG across various DRL subfields and present baseline results using state-of-the-art algorithms. Finally, we conclude by discussing similar benchmarks and related work in section 5, and outline our future plans in section 6.

2 Preliminaries

Markov Decision Processes. In reinforcement learning, an agent interacts with an environment to maximize a cumulative reward signal. This interaction is modelled as a Markov Decision Process (MDP), a 6-tuple $(S, A, r, p, \rho_0, \gamma)$, where S represents the state space, A the action space, $r : S \times A \rightarrow \mathbb{R}$ the reward function, $p : S \times A \times S \rightarrow [0, 1]$ the transition function, $\rho_0 : S \rightarrow [0, 1]$ the initial state distribution, and $\gamma \in [0, 1]$ the discount factor. In the context of PGTG, the agent cannot fully observe the state. Therefore, we consider *partially-observable* MDPs (POMDPs), which extend the MDP by an observation space $\mathcal{O}$ and an observation function $\Omega : S \rightarrow \mathcal{O}$. An agent acts according to a policy $\pi : \mathcal{O} \times A \rightarrow [0, 1]$, where $\pi(a|o)$ denotes the probability of selecting action a given observation o . The optimization objective of RL is to find an optimal policy π^* that maximizes the *expected return* $\mathbb{E}_\pi \left[\sum_{t=0}^T \gamma^t r(s_t, a_t) | s_0 \sim \rho_0 \right]$, for $T \in \mathbb{N} \cup \{\infty\}$.

Gymnasium. Gymnasium (Towers et al., 2024), successor to OpenAI Gym (Brockman et al., 2016), defines a standard interface for abstracting agent-environment interactions. As such, it has been adopted by environments and algorithm frameworks alike (Bellemare et al., 2013; Raffin et al., 2021; Huang et al., 2022; Chevalier-Boisvert et al., 2023). The vast ecosystem of Gymnasium-compliant libraries enables easy incorporation of new tools. By conforming to the Gymnasium API, PGTG ensures interoperability and easy integration into existing research pipelines.

Exploration. The trade-off between exploration and exploitation remains a fundamental challenge in RL. While exploitation leverages current knowledge to maximize returns, an agent must also explore to possibly discover higher-return trajectories. Excessive exploration, however, prevents an agent from learning anything meaningful, as most high-dimensional state spaces cannot possibly be fully explored. Consequently, there is a plethora of work focusing on finding good trade-offs (Ladosz et al., 2022). Many approaches address reward sparsity, where only a few states provide feedback, by augmenting the environment with intrinsic rewards (Aubret et al., 2023) focused on

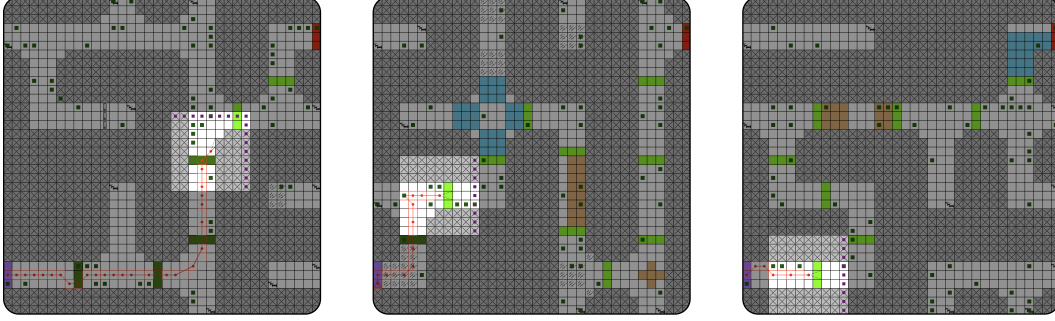


Figure 1: Three generated PGTG tracks rendered in human mode. During generation, the **starting positions** were set to the south-west, and the **goal** to the north-east. The traffic density is rather high (**drivers** as small, green squares) and the **agent's trajectory** is a red connected line, with dots wherever it took an action. Light green lines are **subgoals**, and dark green lines inactive subgoals whose reward was already collected. All obstacles are activated: **ICE** is light blue, **SAND** is orange, **BROKENROAD** is visualized as stripes, and **TRAFFICLIGHTS** show a traffic light symbol. Car symbols represent cells in which new cars can appear. Purple squares at the border of the agent's observation window (the lighter cells) indicate the **navigation system** pointing toward the closest subgoal.

novelty (Burda et al., 2019) or curiosity (Pathak et al., 2017). Orthogonal research investigates how to best define a frontier to decide which states to explore from, irrespective of the underlying exploration strategy (Ecoffet et al., 2021; Pîslar et al., 2022; Gallouédec & Dellandrea, 2023; Duan et al., 2024).

Safety. Safe RL extends the expected return maximization objective by a constraint on an auxiliary cost signal, typically formalized as a Constrained MDP, to bound the expected cost or to ensure that a cost threshold is not violated (García & Fernández, 2015; Zhao et al., 2023). Recently, the importance of safe RL has surged as adhering to safety constraints is often a prerequisite for real-world deployment. Research has focused on *reachability analysis* (Yu et al., 2022; Ganai et al., 2024), *shielding* to block unsafe actions (Alshiekh et al., 2018; Könighofer et al., 2023), or *constrained optimization* to restrict the policy to a feasible region of the parameter space (Achiam et al., 2017).

Generalization. DRL is notoriously sample-inefficient and expensive in comparison to other learning paradigms such as supervised learning (Schwarzer et al., 2020). Given the high computational cost of training, it is indispensable that agents generalize from limited training experience to new, unseen data. For autonomous driving agents, this means generalizing across road features and obstacles rather than overfitting to the layout of a specific training track. Cobbe et al. (2020) point out that using procedural generation is particularly effective to achieve and test generalization.

3 PGTG: Procedurally Generated Grid-Based Traffic Gym

In this section, we provide details on the goals (section 3.1), the features (section 3.2), and the design philosophy (section 3.3) of Procedurally Generated Grid-Based Traffic Gym (PGTG).

3.1 Goals

PGTG provides a unified benchmark to evaluate various DRL challenges in unison or isolation. Its modularity allows users to choose the precise challenges their RL agent should face during training. In pure exploration, measuring state space coverage in sparse-reward environments with high-dimensional spaces is essential. When testing for safety, previous work does not always succeed in decoupling it from exploration, treating cost as negative rewards instead. The result is a sensitive

and crucial parameter that hinders exploration by incentivizing risk-averse convergence to trivial policies, i.e., encouraging the agent to do nothing, conflating the safety and exploration metrics in the empirical study (Ray et al., 2019). To evaluate generalization, the ability to generate sensible out-of-distribution samples at test time is a prerequisite. By isolating these factors, researchers can pinpoint algorithmic weaknesses and identify challenges more accurately.

PGTG addresses these challenges as follows:

1. **Exploration:** PGTG can generate, save, and load arbitrarily complex tasks through the adaptable procedural track generation. By restricting positive rewards to final goal states in a large procedurally generated track, it serves as a robust test environment for sparse reward exploration.
2. **Safety:** By providing an optional navigation system (with subgoals and a subgoal compass), PGTG provides denser rewards to decouple safety from exploration. Including stochastic traffic participants and restricting the agent to partial *local* observability on a procedurally generated track forces agents to generalize over the visual features rather than relying on a global position. Through observable obstacles, the agent can learn to remain safe and avoid uncertainty.
3. **Generalization:** PGTG can be used to evaluate generalization across track sizes or out-of-distribution generation parameters, or for defining a curriculum with increasing track complexity.

3.2 Features

Tracks in PGTG are grid-based and generated through a compositional procedural pipeline using three masks: a wall, an obstacle, and a traffic lane mask. The wall mask defines the navigable space, while the obstacle and traffic masks determine their respective placements (fig. 2). The action space is discrete, comprising nine actions defined by the set $A = \{-1, 0, 1\}^2$. At each time step, the agent selects to accelerate or decelerate independently for both the x and y dimensions, and moves according to its velocity afterward.

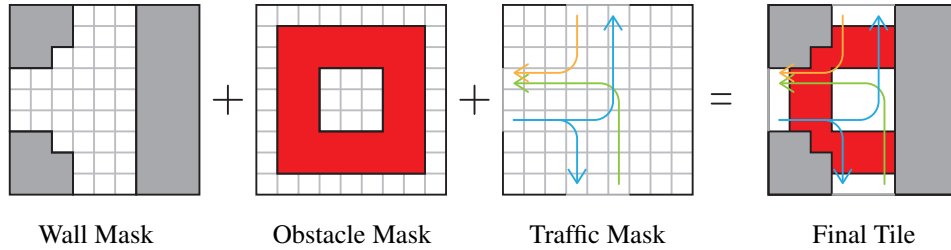


Figure 2: Overlaying the wall, obstacle, and traffic masks yields a generated tile. The obstacle type per position can be predefined or sampled according to a distribution over the supported obstacles.

Drivers. Firstly, we introduce drivers who follow their assigned traffic lane and select a new one at intersections. They keep to the right and have a behavior profile governed by several parameters:

- a probability of whether to stop at a yellow (or red) light,
- a minimal following distance to keep to the car in front,
- a patience level, whether to break any of the above if it has to wait too long,
- probabilities for movement and reaction delays (simulating 1 – 4 steps of inactivity).

An aggressive driver, for instance, might have low patience and a high probability of running red lights. Driver profiles decrease predictability while enabling agents with memory to deduce them.

Obstacles. Secondly, PGTG features obstacles to simulate varying road conditions. Whenever the agent traverses an obstacle, it probabilistically applies its effect, altering the dynamics. Agents can observe obstacles to learn to remain safe and avoid uncertainty. Current obstacles include

- ICE: applies an additional random acceleration,
- BROKENROAD: simulates mechanical failure, resetting the velocity to 0 in all subsequent steps,
- SAND: stops the agent at that position and sets its velocity to 0 this step, and
- TRAFFICLIGHT: imposes a cost penalty when traversed while red.

Traffic Rules. Finally, PGTG features traffic rules defined by preconditions and an effect altering the environment’s dynamics if the preconditions are met. Preconditions consist of (1) the tile’s topology (defined by the connections to adjacent tiles), (2) a velocity range, (3) the direction to the next subgoal, and (4) the local count and directional flow of other drivers within the tile.

3.3 Design Philosophy

The core architecture of PGTG prioritizes customizability and extensibility to support evolving research requirements. Further, with various default obstacles and driver profiles, PGTG keeps the barrier to entry low.

Customizability. PGTG offers a highly modifiable track generation process: each track is generated procedurally from a stochastic adjacency graph defined by grid dimensions and connection density. The graph serves as a blueprint for stitching together predefined tile masks (fig. 2) and supports fixed or random start-goal configurations. The total environment dimensions are a product of the grid and tile resolutions, allowing for significant scale variations. Users can also add novel tiles by providing new tile masks, without modifying the underlying source code. Generated maps can be serialized and loaded to run PGTG in a fixed-map setting.

The observation space is similarly modular to accommodate diverse research objectives. Options include the global coordinates, the local position within a tile, the velocity, and the subgoal’s direction. Central to the agent’s perception is a configurable observation window that can either be a sliding window, or consist of the current tile. It contains one-hot encodings of feature groups. For instance, if the agent observes drivers and goals (= final goal or subgoal), the observation contains two one-hot encoded windows, one per feature group.

Extensibility. PGTG allows for injecting user-defined logic. For example, a new traffic rule can be defined as a json (as seen below) or as a Python dict. Similarly, new driver profiles’ parameters can be passed as a simple dict. Obstacles and traffic rules require additional Python code defining how they alter the dynamics, which is simple, as the ICE example below shows: an obstacle is registered through a decorator without touching the underlying code.

```
1  {
2    "name": "four_way_intersection_brake",
3    "tile_type": [1, 1, 1, 1], // N/E/S/W: 1=connected, 0=not connected
4    "velocity_range": [0.5, 10.0],
5    "min_traffic": 1, // Minimum traffic in tile
6    "min_matching_traffic": 1, // Minimum traffic in tile matching the directions
7    "maneuvers": [ // Agent and traffic directions to match the preconditions
8      { "agent": "west_to_east", "traffic": ["north_to_south", "south_to_north"] },
9      // [...] Other agent / traffic direction matches
10   ], // The action can of course also be a named python function
11   "action": "lambda env, ctx: (setattr(env, 'velocity', np.array((0, 0))),
12               ↳ setattr(ctx, 'acceleration', np.array((0, 0))))"
```

```
1  @pgtg_obstacle("ice", act_probability=0.1, obstacle_weight=1.0, observe=true)
2  def ice_behavior(env: PGTEnv, local_ctx: StepContext) -> bool:
3    ice_action = ice_behavior.rng.choice(len(ACTIONS_TO_ACCELERATION))
4    local_ctx.velocity_substep = ACTIONS_TO_ACCELERATION[ice_action]
5    return False # Additional examples in the supplementary material
```

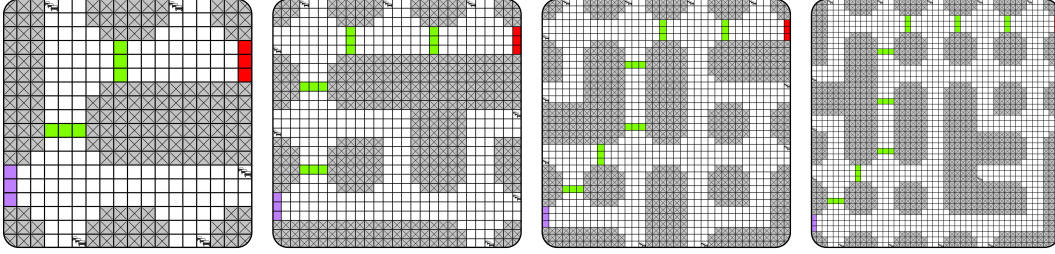


Figure 3: The fixed maps used for the **exploration** experiments.

4 Empirical Evaluation

We conduct a comprehensive suite of experiments evaluating exploration, safety, and generalization of existing state-of-the-art methods on top of DQN (Mnih et al., 2015) and PPO (Schulman et al., 2017). Using WandB (Biewald, 2020), we performed random hyperparameter sweeps spanning 36 configurations for every algorithm-environment pair, optimizing for the primary metric of each task. The following results represent the performance over six independent random seeds using the optimized hyperparameters trained for 10 million time steps on 32 parallel environments. More details on the hyperparameters, as well as training plots, can be found in the supplementary material.

For all experiments, we enable a single traffic rule: It prevents the agent from crossing four-way intersections when there are drivers with orthogonal trajectories, enforcing yielding behavior.

To ensure statistically reliable results, we utilize *deep statistical model checking* (DSMC) (Gros et al., 2020), performing periodic evaluations using PyDSMC (Gros et al., 2025a) (with $\varepsilon = \kappa = 0.05$), and extract the best-performing agent to evaluate it further. For the final reporting, we use tighter bounds with better statistical guarantees in a post-training evaluation (using $\varepsilon = \kappa = 0.01$).

4.1 Exploration

Goal. The objective of exploration is to maximize expected return, which in sparse-reward settings requires first discovering high-return trajectories. Broad state-space coverage is a widely-used heuristic since visiting as much of the state space as possible increases the chance of encountering rewarding regions. Some works even utilize reward-free environments to measure an algorithm’s ability to explore without feedback. Following prior work (Ecoffet et al., 2021; Gallouédec & Delandrea, 2023; Le Tolguenec et al., 2024; Li et al., 2024), we therefore quantify exploration with a coverage proxy: the number of unique, discretized (x, y) cells visited during training.

Settings. To induce reward sparsity, we disable both subgoals and the navigational compass. We utilize fixed tracks of increasing dimensions to ensure the cell-visitation proxy is sensible. A positive reward is only granted upon reaching the track’s final goal, making reward sparsity a function of the track dimensions (see fig. 3). The observation space is restricted to the global and local positions, the velocity, and a sliding window for traffic. To isolate exploration abilities, all obstacles are disabled.

Methods. We benchmark two prominent intrinsic motivation (IM) methods: We evaluate RND (Burda et al., 2019) and ICM (Pathak et al., 2017), as novelty- and prediction-error-based IM, respectively. We use the implementations from RLeXplore (Yuan et al., 2025), and their stable-baselines3 (Raffin et al., 2021) wrappers. Additionally, we include Go-Explore (Ecoffet et al., 2021) and RARE¹ as restoration-based approaches, basing their archives on the (x, y) cell representation.

¹While designed for safety, RARE’s state restorations benefit exploration. Specifically, we evaluate the RARE-ID variant.

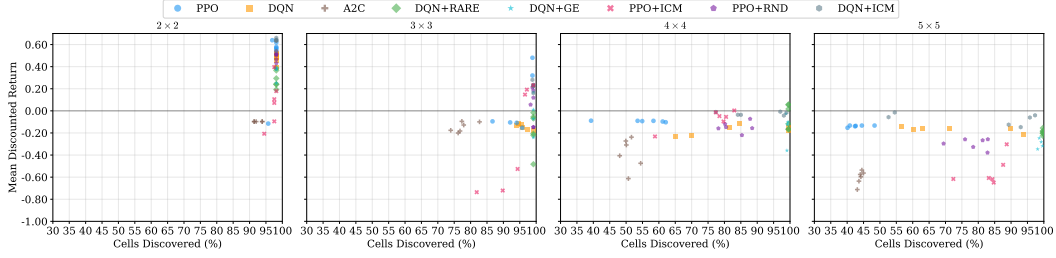


Figure 4: Final results for the **exploration** experiments. Each point represents one seed, evaluated post-training with `PyDSMC` ($\varepsilon = \kappa = 0.01$). We plot cell coverage (x-axis) vs return (y-axis).

Results. Figure 4 features the post-training evaluation results. On the 2×2 track, nearly all algorithms (except for A2C) reliably cover the state space and solve the task. As the track size increases, the task becomes significantly harder. On the 3×3 track, only PPO-based algorithms reach the goal, with PPO+RND doing so most reliably. While other algorithms still cover the state space, they fail to learn a policy solving the task within their 10M time steps training budget. Without any dedicated exploration component, PPO, A2C, and DQN all struggle to cover the state space on larger tracks, with DQN exhibiting a high variance, particularly pronounced on the 5×5 track—though augmenting it with ICM helps tremendously. The tasks are challenging even for state-of-the-art algorithms, as is evident from the low state space coverage and negative return on the large tracks.

4.2 Safety

Goal. The focus in the safety experiments is to evaluate an agent’s ability to maximize returns while adhering to safety constraints, such as avoiding collisions and stopping at red lights. Unlike the exploration tasks, we are interested in constraint satisfaction and risk-averse navigation in the presence of informative feedback.

Settings. To provide a dense learning signal, subgoals and a directional navigation compass are enabled. Additionally, we enable all supported obstacles and procedurally generate a new track each episode. The agent cannot observe its global position and has a separate observation window for each obstacle type. We formalize this task as a Constrained MDP, where the cost signal (e.g., on crashes) is separated from the rewards (on subgoals).

Methods. We evaluate Constrained Policy Optimization (CPO) (Achiam et al., 2017), PPO-Lag (Ray et al., 2019), and RARE² (Gros et al., 2025b). The former two enforce a cost threshold ($= 0.5$) via constrained optimization or Lagrangian relaxation, respectively. The latter employs state restorations, i.e., restarts from previously visited states, to focus learning on states whose estimated regret is high using a singular signal: $\text{reward} - \text{cost}$. For CPO and PPO-Lag, we use the implementations from Ji et al. (2023), whereas we reimplement RARE because the original implementation is not readily applicable to new environments.

Results. Figure 5 shows the evaluation results and illustrates the return-cost trade-off. Since the total achievable return per episode differs due to the procedural generation, we report the subgoal completion rate as a normalized metric. On 2×2 tracks, CPO maintains a superior safety profile without sacrificing completion rate. However, as the grid scales, CPO becomes increasingly conservative, leading to a noticeable decline on larger tracks. The relaxed objective of PPO-Lag does not seem beneficial in this setting. Notably, there is a large improvement potential as all the algorithms struggle to reach a subgoal completion rate approaching 1 while staying safe.

²We only consider the version denoted RARE-ID in the original paper.

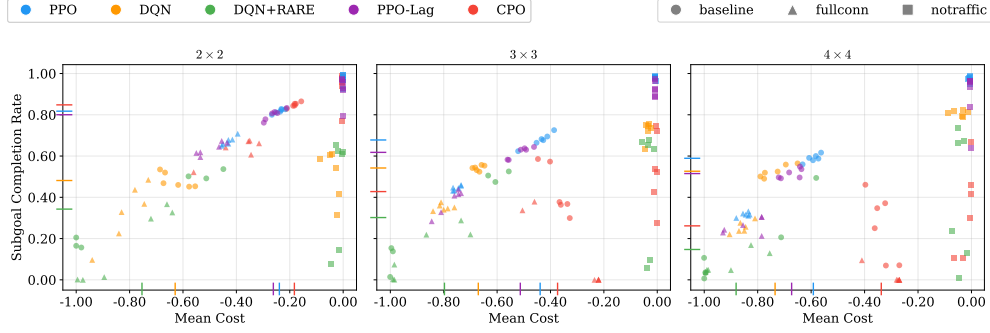


Figure 5: Results for **safety**: Each point represents one seed, evaluated post-training with `PyDSMC` ($\varepsilon = \kappa = 0.01$). The colored axis ticks represent the mean values of the baselines’ evaluations.

Additionally, we evaluate the policies out-of-distribution on fully-connected tracks and ones without traffic. In the fully connected setting, performance degrades, as expected, for two reasons: The policies were not trained in this setting, and traffic can come from all directions at all intersections. Even though the policies have never seen tracks without traffic, they have not overly relied on spurious correlations of other drivers being present, as indicated by the higher completion rates.

4.3 Generalization

Goal. For generalization, an agent ought to learn to apply its knowledge to previously unseen situations. In consequence, to be optimal, the agent has to learn the true connections between observations as spurious correlations or shortcuts (Geirhos et al., 2020; Suau et al., 2024) hurt performance.

Settings. The configuration for generalization follows the safety setup, with two primary distinctions: (1) the reward signal is monolithic, incorporating cost as a negative penalty, and (2) the evaluation focuses on distributional shift and environmental scaling. We train agents on various track dimensions, and evaluate them on different, unseen ones.

Methods. For generalization, we compare vanilla versions of DQN, PPO, and A2C only. Since rewards are dense, these should still perform reasonably well, and our goal is to showcase PGTG’s capabilities to test generalization, not to find the best generalization method.

Results. Figure 6 summarizes the agents’ transfer performance across track dimensions. Because of the restricted local observability, we expect the performance to be rather invariant to the track dimensions. Nevertheless, the small number of possible 2×2 tracks is not representative of larger ones, causing a strong performance degradation. In contrast, policies trained on 3×3 tracks generalize better to the 4×4 ones, but struggle as the scale increases further. Agents trained on 4×4 exhibit the most robust performance transfer to the 5×5 and 6×6 tracks. Crucially, no algorithm reaches perfect completion rates or flat trends across track dimensions, highlighting the relevance of this setting for future research.

5 Related Work

Simulation benchmarks, and in particular abstractions of autonomous vehicles, have been used extensively in the past. One of the most prominent examples is RACETRACK (Sutton & Barto, 2018; Baier et al., 2021), which requires an agent to navigate a grid-based environment to a goal line while managing momentum under stochastic action noise. As explained in section 1, RACETRACK’s utility is limited by its simplicity. While a degenerate configuration of PGTG (on a fixed map, with all features disabled) can emulate RACETRACK, PGTG’s modularity, customizability, and extensibility make it a complex and relevant research-oriented benchmark.

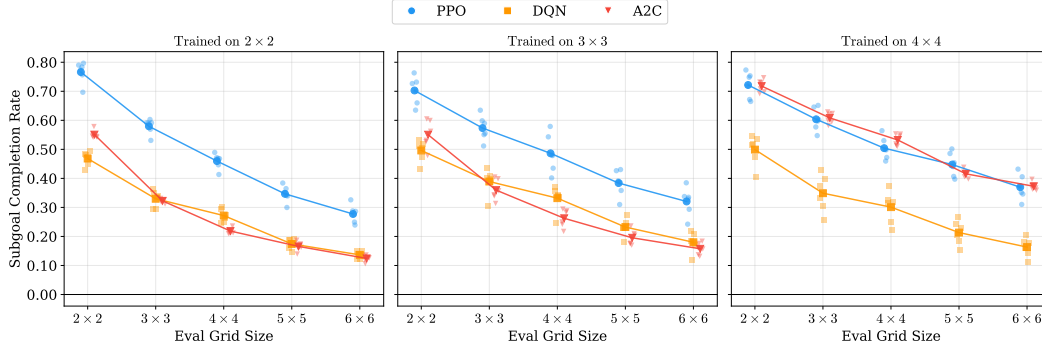


Figure 6: Results for **generalization**: Each small point represents one seed, evaluated post-training with `PyDSMC` ($\epsilon = \kappa = 0.01$). The connected opaque points correspond to the mean values.

Another widely adopted grid-based framework is MINIGRID (Chevalier-Boisvert et al., 2023). In lieu of a vehicle with momentum, MINIGRID agents move deterministically through the grid one cell at a time. This fundamental difference turns MINIGRID into a high-level planning task, while PGTG incorporates low-level control and high-level planning aspects. MINIGRID features procedurally generated maps that, like PGTG, can be used for curriculum learning (Portelas et al., 2020; Dennis et al., 2020; Campero et al., 2021), and to evaluate exploration and generalization (Raileanu & Rocktäschel, 2020; Zhang et al., 2021b;a; Wang et al., 2025). Customization in MINIGRID, however, requires full Python-level implementations, which are not only more complex, but also lead to API fragmentation and complicate efforts to implement suite-wide features like restoration capabilities.

Beyond grid-based domains, rigid-body physics engines such as MUJoCo (Todorov et al., 2012) and its derivatives, e.g., SAFETYGYM (Ray et al., 2019; Ji et al., 2023) and ANTMAZE (de Lazcano et al., 2024), are commonly used to benchmark safety and exploration (Achiam et al., 2017; Fu et al., 2021; Li et al., 2023; Mohamed et al., 2025). While complex, high-fidelity simulations are powerful for integration and stress testing, their high-dimensional state and action spaces introduce confounding complexities. This makes it significantly harder to isolate specific failure points and draw clear conclusions. As physics simulation engines, they are also quite heavy and not easily extensible.

Finally, drone racing environments (Song et al., 2021a;b) serve as popular benchmarks for exploration and safety in control. They often employ a reward structure similar to PGTG, incentivizing agents to pass through sequential gates using compass-based navigation (Kaufmann et al., 2023).

6 Conclusion

PGTG is a highly customizable and extensible Gymnasium-compliant benchmark for reinforcement learning. Through its modular architecture and configurable features, researchers can effectively isolate and investigate specific algorithmic challenges and analyze fundamental trade-offs, such as the tension between return and safety. Our empirical study demonstrates that PGTG is a versatile benchmark capable of providing insights across various deep RL subfields, such as exploration, safety, and generalization. Crucially, no tested state-of-the-art algorithm achieves satisfactory performance. We therefore anticipate that this framework will foster new research insights, and we intend to continuously extend it to support a wider array of obstacles, tiles, and navigational challenges. For instance, currently, the tile design does not support diagonal connections, as generating these requires a more complex interplay between adjacent tiles. Moreover, we plan support for continuous state and action spaces, to extend the utility of PGTG to more complex optimization landscapes (Duan et al., 2016) and to algorithms specifically tailored to continuous control (Haarnoja et al., 2018).

Acknowledgments

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – GRK 2853/1 “Neuroexplicit Models of Language, Vision, and Action” – project number 471607914, and under Grant No. 389792660, as part of TRR 248 – CPEC, see <https://perspicuous-computing.science>, by the German Federal Ministry of Education and Research (BMBF) as part of project MAC-MERLin (grant agreement no. 01IW24007), and by the European Regional Development Fund (ERDF) and Saarland within the scope of (To)CERTAIN.

Generative AI Disclosure

The authors used generative AI models, specifically large language models (LLMs), as an assistive tool during the preparation of this work. LLMs were used to refine the phrasing of author-written text and to assist with code review, including identifying potential bugs. Generative AI was not used to independently produce sections of the paper or code. All AI-assisted output was reviewed and verified by the authors, who take full responsibility for the correctness of the contents presented here.

References

- Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained Policy Optimization. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 22–31. PMLR, July 2017.
- Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe Reinforcement Learning via Shielding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2374-3468. DOI: 10.1609/aaai.v32i1.11797.
- Arthur Aubret, Laetitia Matignon, and Salima Hassas. An Information-Theoretic Perspective on Intrinsic Motivation in Reinforcement Learning: A Survey. *Entropy*, 25(2):327, February 2023. ISSN 1099-4300. DOI: 10.3390/e25020327.
- Christel Baier, Maria Christakis, Timo P. Gros, David Groß, Stefan Gumhold, Holger Hermanns, Jörg Hoffmann, and Michaela Klauk. Lab Conditions for Research on Explainable Automated Decisions. In *Trustworthy AI - Integrating Learning, Optimization and Reasoning*, pp. 83–90, Cham, 2021. Springer International Publishing. ISBN 978-3-030-73959-1. DOI: 10.1007/978-3-030-73959-1_8.
- M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013. ISSN 1076-9757. DOI: 10.1613/jair.3912.
- Lukas Biewald. Experiment tracking with weights and biases, 2020. URL <https://www.wandb.com/>. Software available from wandb.com.
- Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, June 2016.
- Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *7th International Conference on Learning Representations (ICLR 2019)*, pp. 1–17, May 2019. DOI: 10.48550/arxiv.1810.12894.
- Andres Campero, Roberta Raileanu, Heinrich Küttler, Joshua B. Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. Learning with AMIGO: Adversarially Motivated Intrinsic Goals, February 2021.

- Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and J. Terry. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. *Advances in Neural Information Processing Systems*, 36:73383–73394, December 2023. DOI: 10.48550/arxiv.2306.13831.
- Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging Procedural Generation to Benchmark Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 2048–2056. PMLR, November 2020.
- Pim de Haan, Dinesh Jayaraman, and Sergey Levine. Causal Confusion in Imitation Learning. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- Rodrigo de Lazcano, Kallinteris Andreas, Jun Jet Tai, Seungjae Ryan Lee, and Jordan Terry. Gymnasium robotics, 2024. URL <http://github.com/Farama-Foundation/Gymnasium-Robotics>.
- Quentin Delfosse, Sebastian Sztiwierz, Mark Rothermel, Wolfgang Stammer, and Kristian Kersting. Interpretable Concept Bottlenecks to Align Reinforcement Learning Agents. *Advances in Neural Information Processing Systems*, 37:66826–66855, December 2024.
- Michael Dennis, Natasha Jaques, Eugene Vinitzky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13049–13061. Curran Associates, Inc., 2020. DOI: 10.48550/arxiv.2012.02096.
- Yan Duan, Xi Chen, Rein Houthoofd, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1329–1338, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <https://proceedings.mlr.press/v48/duan16.html>.
- Yuanlin Duan, Guofeng Cui, and He Zhu. Exploring the Edges of Latent State Clusters for Goal-Conditioned Reinforcement Learning, November 2024.
- Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, February 2021. ISSN 1476-4687. DOI: 10.1038/s41586-020-03157-9.
- Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, October 2022. ISSN 1476-4687. DOI: 10.1038/s41586-022-05172-4.
- Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for Deep Data-Driven Reinforcement Learning, February 2021.
- Quentin Gallouédec and Emmanuel Dellandrea. Cell-Free Latent Go-Explore. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 10571–10586. PMLR, July 2023.
- Milan Ganai, Sicun Gao, and Sylvia L. Herbert. Hamilton-Jacobi Reachability in Reinforcement Learning: A Survey. *IEEE Open Journal of Control Systems*, 3:310–324, 2024. ISSN 2694-085X. DOI: 10.1109/OJCSYS.2024.3449138.
- Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *The Journal of Machine Learning Research*, 16(1):1437–1480, January 2015. ISSN 1532-4435.
- Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, November 2020. ISSN 2522-5839. DOI: 10.1038/s42256-020-00257-z.

- Timo P. Gros, Holger Hermanns, Jörg Hoffmann, Michaela Klauck, and Marcel Steinmetz. Deep Statistical Model Checking. In *Formal Techniques for Distributed Objects, Components, and Systems*, pp. 96–114, Cham, 2020. Springer International Publishing. ISBN 978-3-030-50086-3. DOI: 10.1007/978-3-030-50086-3_6.
- Timo P. Gros, Arnd Hartmanns, Ivo Hoes, Joshua Meyer, Nicola J. Müller, and Verena Wolf. PyDSMC: Statistical Model Checking for Neural Agents Using the Gymnasium Interface. In *Quantitative Evaluation of Systems and Formal Modeling and Analysis of Timed Systems*, pp. 134–156, Cham, 2025a. Springer Nature Switzerland. ISBN 978-3-032-05792-1. DOI: 10.1007/978-3-032-05792-1_8.
- Timo P. Gros, Nicola J. Müller, Daniel Höller, and Verena Wolf. Safe Reinforcement Learning Through Regret and State Restorations in Evaluation Stages. In *Principles of Verification: Cycling the Probabilistic Landscape: Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday, Part III*, pp. 18–38. Springer Nature Switzerland, Cham, 2025b. ISBN 978-3-031-75778-5. DOI: 10.1007/978-3-031-75778-5_2.
- Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1861–1870. PMLR, July 2018.
- Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. ISSN 1533-7928.
- Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, pp. 1–3, November 2025. ISSN 1476-4687. DOI: 10.1038/s41586-025-09833-y.
- Jiaming Ji, Borong Zhang, Jiayi Zhou, Xuehai Pan, Weidong Huang, Ruiyang Sun, Yiran Geng, Yifan Zhong, Juntao Dai, and Yaodong Yang. Safety-gymnasium: A unified safe reinforcement learning benchmark. *arXiv preprint arXiv:2310.12567*, 2023. URL <https://github.com/PKU-Alignment/Safe-Policy-Optimization>.
- Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, August 2023. ISSN 1476-4687. DOI: 10.1038/s41586-023-06419-4.
- Bettina Könighofer, Julian Rudolf, Alexander Palmisano, Martin Tappler, and Roderick Bloem. Online shielding for reinforcement learning. *Innovations in Systems and Software Engineering*, 19(4):379–394, December 2023. ISSN 1614-5054. DOI: 10.1007/s11334-022-00480-4.
- Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85:1–22, September 2022. ISSN 1566-2535. DOI: 10.1016/j.inffus.2022.03.003.
- Lauro Langosco Di Langosco, Jack Koch, Lee D. Sharkey, Jacob Pfau, and David Krueger. Goal Misgeneralization in Deep Reinforcement Learning. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 12004–12019. PMLR, June 2022.
- Paul-Antoine Le Tolguenec, Yann Besse, Florent Teichteil-Konigsbuch, Dennis G. Wilson, and Emmanuel Rachelson. Exploration by Learning Diverse Skills through Successor State Representations. *Advances in Neural Information Processing Systems*, 37:75277–75301, December 2024. DOI: 10.52202/079017-2396.

- Qiyang Li, Jason Zhang, Dibya Ghosh, Amy Zhang, and Sergey Levine. Accelerating Exploration with Unlabeled Prior Data. *Advances in Neural Information Processing Systems*, 36:67434–67458, December 2023. DOI: 10.48550/arxiv.2311.05067.
- Zechu Li, Rickmer Krohn, Tao Chen, Anurag Ajay, Pulkit Agrawal, and Georgia Chalvatzaki. Learning Multimodal Behaviors from Scratch with Diffusion Policy Gradient. *Advances in Neural Information Processing Systems*, 37:38456–38479, December 2024.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Belle-mare, Alex Graves, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, February 2015. ISSN 1476-4687. DOI: 10.1038/nature14236.
- Faisal Mohamed, Catherine Ji, Benjamin Eysenbach, and Glen Berseth. Curiosity-Driven Exploration via Temporal Contrastive Learning. In *Workshop on Reinforcement Learning Beyond Rewards @ Reinforcement Learning Conference 2025*, July 2025.
- Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven Exploration by Self-supervised Prediction. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2778–2787. PMLR, July 2017. DOI: 10.1109/CVPRW.2017.70.
- Miruna Pislar, David Szepesvari, Georg Ostrovski, Diana Borsa, and Tom Schaul. When should agents explore?, March 2022.
- R  my Portelas, C  dric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments. In *Proceedings of the Conference on Robot Learning*, pp. 835–853. PMLR, May 2020.
- Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dornmann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. ISSN 1533-7928.
- Roberta Raileanu and Tim Rock  schel. RIDE: Rewarding Impact-Driven Exploration for Procedurally-Generated Environments, February 2020.
- Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking Safe Exploration in Deep Reinforcement Learning. November 2019.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017.
- Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron Courville, and Philip Bachman. Data-Efficient Reinforcement Learning with Self-Predictive Representations. In *International Conference on Learning Representations*, October 2020.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, December 2018. DOI: 10.1126/science.aar6404.
- Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A Flexible Quadrotor Simulator. In *Proceedings of the 2020 Conference on Robot Learning*, pp. 1147–1157. PMLR, October 2021a.
- Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous Drone Racing with Deep Reinforcement Learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1205–1212, September 2021b. DOI: 10.1109/IROS51168.2021.9636053.
- Miguel Suau, Matthijs T J Spaan, and Frans A Oliehhoek. Bad Habits: Policy Confounding and Out-of-Trajectory Generalization in RL. 4:1711–1732, 2024.

- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, Second Edition: An Introduction*. MIT Press, November 2018. ISBN 978-0-262-35270-3. DOI: 10.1109/tnn.2004.842673.
- Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, October 2012. DOI: 10.1109/IROS.2012.6386109.
- Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. *Gymnasium: A Standard Interface for Reinforcement Learning Environments*, November 2024.
- Hongze Wang, Jiaxu Xing, Nico Messikommer, and Davide Scaramuzza. Environment as Policy: Learning to Race in Unseen Tracks. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11333–11339, May 2025. DOI: 10.1109/ICRA55743.2025.11127376.
- Dongjie Yu, Haitong Ma, Shengbo Li, and Jianyu Chen. Reachability Constrained Reinforcement Learning. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 25636–25655. PMLR, June 2022.
- Mingqi Yuan, Roger Creus Castanyer, Bo Li, Xin Jin, Glen Berseth, and Wenjun Zeng. Rlexplore: Accelerating research in intrinsically-motivated reinforcement learning. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL <https://openreview.net/forum?id=B9BHjTN4z6>.
- Hongming Zhang and Tianyang Yu. AlphaZero. In *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pp. 391–415. Springer, Singapore, 2020. ISBN 978-981-15-4095-0. DOI: 10.1007/978-981-15-4095-0_15.
- Tianjun Zhang, Paria Rashidinejad, Jiantao Jiao, Yuandong Tian, Joseph E Gonzalez, and Stuart Russell. MADE: Exploration via Maximizing Deviation from Explored Regions. In *Advances in Neural Information Processing Systems*, volume 34, pp. 9663–9680. Curran Associates, Inc., 2021a. DOI: 10.48550/arxiv.2106.10268.
- Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. NovelD: A Simple yet Effective Exploration Criterion. In *Advances in Neural Information Processing Systems*, volume 34, pp. 25217–25230. Curran Associates, Inc., 2021b.
- Weiye Zhao, Tairan He, Rui Chen, Tianhao Wei, and Changliu Liu. State-wise Safe Reinforcement Learning: A Survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 6814–6822, Macau, SAR China, August 2023. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-1-956792-03-4. DOI: 10.24963/ijcai.2023/763.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Hyperparameter Configurations

Table 1: Hyperparameter configuration for the **safety** scenario across different track sizes. Unmentioned parameters are left at their default values (stable-baselines3, version 2.7.0).

Algorithm	Hyperparameter	2×2	3×3	4×4
PPO	ent_coef	0.05	0.05	0.05
	learning_rate	$7e-4$	$3e-4$	$3e-4$
	n_steps	2048	2048	2048
	vf_coef	0.2	0.5	0.5
	batch_size	512	512	512
DQN	buffer_size	1 000 000	1 000 000	1 000 000
	gradient_steps	32	16	16
	learning_rate	$1e-4$	$1e-4$	$1e-4$
	target_update_interval	32	32	128
	tau	0.005	0.005	0.1
	train_freq	1	1	1
RARE	buffer_size	1 000 000	1 000 000	10 000 000
	gradient_steps	32	32	16
	learning_rate	$1e-4$	$1e-4$	$1e-4$
	target_update_interval	32	32	32
	tau	0.005	0.05	0.05
	train_freq	1	1	1
	eval_alpha	1.0	1.0	1.0
	eval_stage_freq	500 000	500 000	500 000
PPO-Lag	minimal_priority	0.2	0.2	0.2
	ent_coef	0.05	0.05	0.05
	learning_rate	$7e-4$	$3e-4$	$3e-4$
	n_steps	2048	2048	2048
	vf_coef	0.2	0.5	0.5
	batch_size	512	512	512
	cost_limit	0.5	0.5	0.5
	lagrange_init	$1e-3$	$1e-3$	$1e-3$
CPO	lagrange_lr	$5e-2$	$5e-2$	$5e-2$
	learning_rate	$7e-4$	$3e-4$	$3e-4$
	vf_coef	0.2	0.5	0.5
	n_steps	2048	2048	2048
	batch_size	2048	2048	2048
	ent_coef	0.01	0.01	0.01
	cost_limit	0.5	0.5	0.5
	lagrange_lr	$5e-2$	$5e-2$	$5e-2$
	lagrange_init	$1e-3$	$1e-3$	$1e-3$
	cpo_target_kl	0.01	0.01	0.01
	cost_vf_coef	0.5	0.5	0.5

Table 2: Hyperparameter configuration for the **exploration** scenario across different PGTG map sizes. Unmentioned parameters are left at their default values (stable-baselines3, version 2.7.0).

Algorithm	Hyperparameter	2×2	3×3	4×4	5×5
A2C	ent_coef	0.001	0.001	0.05	0.05
	learning_rate	$7e-4$	$7e-4$	$7e-4$	$7e-4$
	n_steps	2048	2048	2048	2048
	vf_coef	0.2	0.2	0.2	0.2
DQN	buffer_size	1 000 000	1 000 000	1 000 000	1 000 000
	gradient_steps	32	32	16	16
	learning_rate	$1e-4$	$1e-3$	$8e-4$	$8e-4$
	target_update_interval	128	128	128	128
	tau	0.005	0.005	0.005	0.005
	train_freq	1	1	1	1
PPO	ent_coef	0.001	0.001	0.05	0.05
	learning_rate	$7e-4$	$7e-4$	$7e-4$	$7e-4$
	n_steps	2048	2048	2048	2048
	vf_coef	0.2	0.2	0.2	0.2
	batch_size	512	512	512	512
PPO+RND	latent_size	128	64	64	64
	beta	1	1	1	10
	vf_coef	0.5	0.5	0.5	0.5
	n_steps	2048	256	256	256
	learning_rate	$3e-4$	$3e-4$	$3e-4$	$3e-4$
	ent_coef	0.05	0.001	0.001	0.001
DQN+ICM	batch_size	64	256	256	256
	buffer_size	1 000 000	1 000 000	1 000 000	1 000 000
	gradient_steps	32	32	32	32
	learning_rate	$1e-4$	$1e-4$	$1e-4$	$1e-4$
	target_update_interval	32	32	32	32
	tau	0.05	0.05	0.05	0.05
	beta	0.1	0.1	1	0.1
	latent_size	64	64	64	64
PPO+ICM	learning_rate_icm	$2e-4$	$2e-4$	$2e-4$	$2e-4$
	batch_size	1024	1024	1024	1024
	ent_coef	0.005	0.005	0.005	0.005
	learning_rate	$3e-4$	$3e-4$	$3e-4$	$3e-4$
	n_steps	256	256	256	256
	vf_coef	0.5	0.5	0.5	0.5
	beta	0.1	1	1	1
	latent_size	64	64	64	64
	learning_rate_icm	$2e-4$	$1e-4$	$1e-4$	$1e-4$

Table 3: Hyperparameter configuration for the **generalization** scenario across different track sizes. Unmentioned parameters are left at their default values (stable-baselines3, version 2.7.0).

Algorithm	Hyperparameter	2×2	3×3	4×4
PPO	ent_coef	0.001	0.001	0.05
	learning_rate	$3e-4$	$3e-4$	$3e-4$
	n_steps	2048	2048	2048
	vf_coef	0.5	0.2	0.5
	batch_size	512	64	64
DQN	buffer_size	1 000 000	1 000 000	1 000 000
	gradient_steps	16	16	16
	learning_rate	$1e-4$	$1e-4$	$1e-4$
	target_update_interval	32	128	32
	tau	0.05	0.05	0.05
	train_freq	1	1	1
A2C	ent_coef	0.05	0.05	0.05
	learning_rate	$7e-4$	$7e-4$	$7e-4$
	n_steps	2048	2048	5
	gae_lambda	1.0	1.0	1.0
	vf_coef	0.5	0.5	0.5

Table 4: DSMC configuration during training for **exploration**, **safety** and **generalization**.

Parameters that are not repeated remain the same as in the previous setting.

For the post-training evaluation, we set $\varepsilon = \kappa = 0.01$.

Setting	Parameter	Value
Exploration	interval (time steps)	100 000
	first evaluation (time steps)	100 000
	episodes per policy run	50
	initial episodes	50
	episode time limit (minutes)	3
	deterministic	true
	confidence parameter (κ)	0.05
	error tolerance (ε)	0.05
	discount factor (γ)	0.99
	zero-variance cutoff (episodes)	200
Safety	relative error	false
	interval (time steps)	500 000
Generalization	first evaluation (time steps)	500 000
	interval (time steps)	200 000
Generalization	first evaluation (time steps)	200 000

B Hyperparameter Search Spaces

Table 5: Baseline hyperparameters analyzed with a sweep of 36 runs on all environments. All methods building upon a particular base algorithm also additionally analyzed these parameters.

Method	Hyperparameter	Search Space
DQN	learning rate	$\{8e-4, 1e-4\}$
	buffer size	$\{1e7, 1e6\}$
	soft update τ	$\{0.1, 0.05, 0.005\}$
	gradient steps	$\{16, 32\}$
	target update frequency	$\{1, 128\}$
PPO	learning rate	$\{7e-4, 3e-4, 7e-5\}$
	batch size	$\{64, 512\}$
	train frequency (steps)	$\{256, 2048\}$
	entropy coefficient	$\{5e-2, 1e-4\}$
	value function coefficient	$\{0.2, 0.5\}$
A2C	learning rate	$\{7e-4, 3e-4, 7e-5\}$
	train frequency (steps)	$\{5, 256, 2048\}$
	entropy coefficient	$\{5e-2, 1e-3\}$
	value function coefficient	$\{0.2, 0.5\}$

Table 6: Hyperparameter search spaces for all specialized algorithms. The search spaces are joined with the baseline’s search space provided in table 5 to comprise a full sweep configuration.

Method	Hyperparameter	Search Space
RND	intrinsic reward coeff. β	$\{0.001, 0.01, 0.1, 1, 10\}$
	update proportion	$\{0.25, 0.5, 1\}$
	latent embedding size	$\{64, 128\}$
	reward normalization	$\{\text{rms}, \text{none}\}$
ICM	intrinsic reward coeff. β	$\{0.01, 0.1, 1, 10, 100\}$
	latent embedding size	$\{16, 64, 128\}$
	ICM learning rate	$[1e-5, 1e-3]$
	reward normalization	$\{\text{rms}, \text{none}\}$
RARE	eval alpha	$\{0.5, 1\}$
	eval stage frequency	$\{1e5, 2e5, 4e5\}$
	minimal priority	$\{0.1, 0.2\}$
PPO-Lag & CPO	KL target	$\{0.005, 0.01, 0.02\}$
	Lagrange init	$\{0.001, 0.01, 0.1\}$
	Lagrange learning rate	$\{0.01, 0.05, 0.1\}$

C Environment Arguments

Table 7: Environment configuration for the PGTG benchmark. Scenario columns show overrides relative to the base configuration. The agent’s observation features contain each generable obstacle as an observation channel. In Exploration, the agent is unable to see subgoals, while in safety and generalization, the agent observes `goals`; i.e., it cannot differentiate between subgoals and the final goal. Moreover, since in exploration we utilize fixed maps, the map context describes the parameters used to generate the fixed map.

Parameter	Exploration	Safety	Generalization
<i>General</i>			
traffic_density	0.1	0.2	0.2
traffic_light_phases	[10, 3, 10]	[10, 3, 10]	[10, 3, 10]
ignore_traffic_collisions	false	false	false
<i>Observation Context</i>			
window_size	9	9	9
subgoal_direction	false	true	true
global_pos	true	false	false
local_pos	true	true	true
velocity	true	true	true
<i>Map Context</i>			
map_size	$[2 \times 2 - 5 \times 5]$	$[2 \times 2 - 4 \times 4]$	$[2 \times 2 - 6 \times 6]$
connections_percentage	0.8	0.5	0.5
obstacle_probability	0.0	0.2	0.2
obstacle_weight: ice	1	1	1
obstacle_weight: broken_road	1	1	1
obstacle_weight: sand	1	1	1
obstacle_weight: traffic_light	1	1	1
<i>Reward Context</i>			
subgoal_reward	0.0	1.0	1.0
final_goal_bonus	1.0	0.0	0.0
crash_penalty	1.0	1.0	1.0
traffic_light_penalty	0.05	0.05	0.05
standing_still_penalty	0.1	0	0
split_subgoal_reward	false	false	false
separate_reward_cost	false	true	false
<i>Driver Profile Weights</i>			
normal	1.0	0.8	0.8
elderly	0.0	0.2	0.2
aggressive	0.0	0.0	0.0
reckless	0.0	0.0	0.0
conservative	0.0	0.0	0.0

C.1 Description of Environment Arguments

PGTG exposes a comprehensive set of configuration parameters that allow researchers to tailor the environment to specific experimental needs. These parameters span traffic simulation, observation composition, procedural map generation, reward shaping, and driver-profile sampling. By selectively adjusting these settings, distinct scenario profiles can be instantiated from a single code-base—for instance, toggling between sparse exploration rewards and dense safety-constrained signals without modifying the underlying environment logic. This not only allows isolating only specific challenges, but also the trade-offs between complementary objectives.

General Settings. The *General* configuration governs the core simulation dynamics. `traffic_density` specifies the fraction of traffic-eligible squares that are occupied (1.0 means all lanes are permanently filled) at each time step. New cars appear if the specified density is not met. `traffic_light_phases` is the duration of each traffic light phase (green, yellow, red) in steps. `ignore_traffic_collisions` disables termination on a collision with traffic.

Observation Context. The *Observation Context* parameters control what information the agent receives. Observations are constructed from a configurable set of features that can be independently toggled, allowing for controlled partial observability. `window_size` controls the dimensions of the one-hot encoded feature windows (e.g., 9 corresponds to a 9×9 window). `subgoal_direction` provides an implicit navigation hint, in the form of a direction to the closest subgoal. This reduces the challenge of learning where to navigate and allows the agent to focus on safe driving behavior. `global_pos` provides absolute map coordinates, whereas `local_pos` only provides the relative position within the current tile. `velocity` exposes the current velocity vector, enabling more informed acceleration and braking decisions.

Map Context. The *Map Context* parameters govern procedural and non-procedural grid generation. Each episode can be initialized on a newly sampled map. Adjusting `map_size` scales the navigable area in tile increments. Larger maps introduce more subgoals and longer episodes without altering the local difficulty of reaching individual waypoints. Without subgoals, however, larger tracks increase reward sparsity tremendously. `connections_percentage` controls how densely tiles are connected, i.e., how many of the possible four-directional connections are generated. `obstacle_probability` sets the probability that a tile contains obstacles. The probability of each obstacle type is determined by normalizing the obstacle weights of all enabled obstacles.

Reward Context. The *Reward Context* allows tuning the magnitude and composition of rewards. When `split_subgoal_reward` is enabled, a fixed total reward is distributed evenly across all subgoals; otherwise, each subgoal yields a constant reward independent of the total number of subgoals. `separate_reward_cost` splits the return into a positive performance reward and a non-negative safety cost (e.g., crashes or red-light violations), enabling constrained optimization and multi-objective RL. The `final_goal_bonus` is granted once the last subgoal is reached, and given on top of the `subgoal_reward`, i.e., the total obtained reward is `subgoal_reward + final_goal_bonus`. When both acceleration and velocity are 0, the `standing_still_penalty` is applied.

Drivers. PGTG models traffic participants with five distinct driver profiles: *conservative*, *normal*, *aggressive*, *elderly*, and *reckless*. Each profile’s parameters govern its behavior: yellow-light stopping probability, red-light violation probability, minimum following distance, patience level, speed multiplier, and reaction delay probability. Driver profile weights are normalized to a probability distribution to assign profiles to new drivers probabilistically. By adjusting these weights, the overall traffic behavior can be shifted from predominantly cautious to predominantly aggressive, enabling the study of how surrounding traffic composition affects the learning of safe driving policies.

D Training Curves

In the following, we provide train curves for all experiments and settings. The shaded area corresponds to a 95% confidence interval around the mean in all cases.

D.1 Exploration

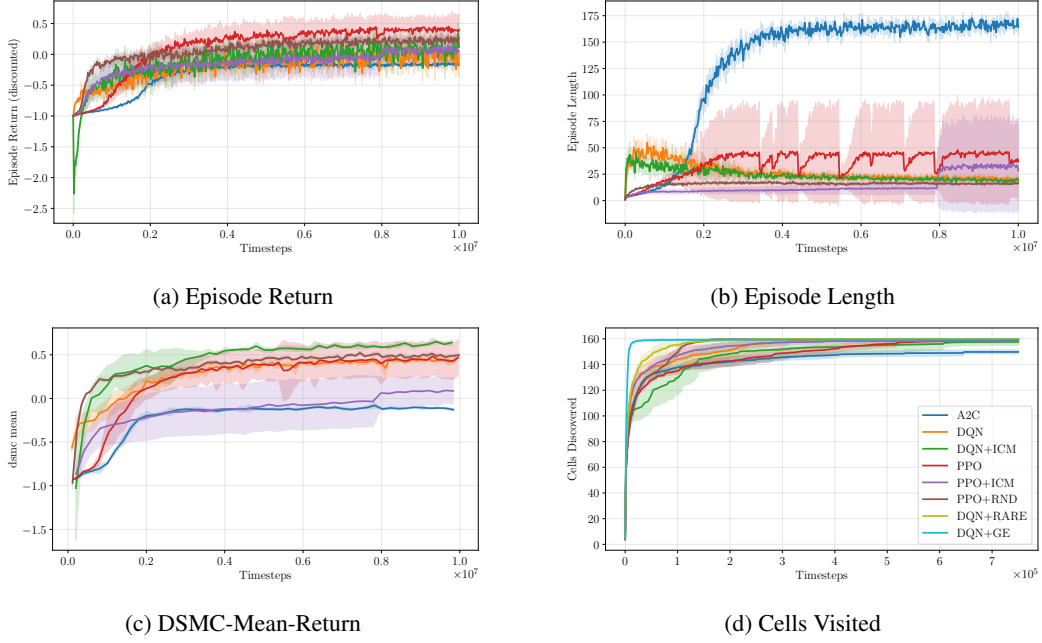


Figure 7: Training curves for the **exploration** scenario on 2×2 maps.

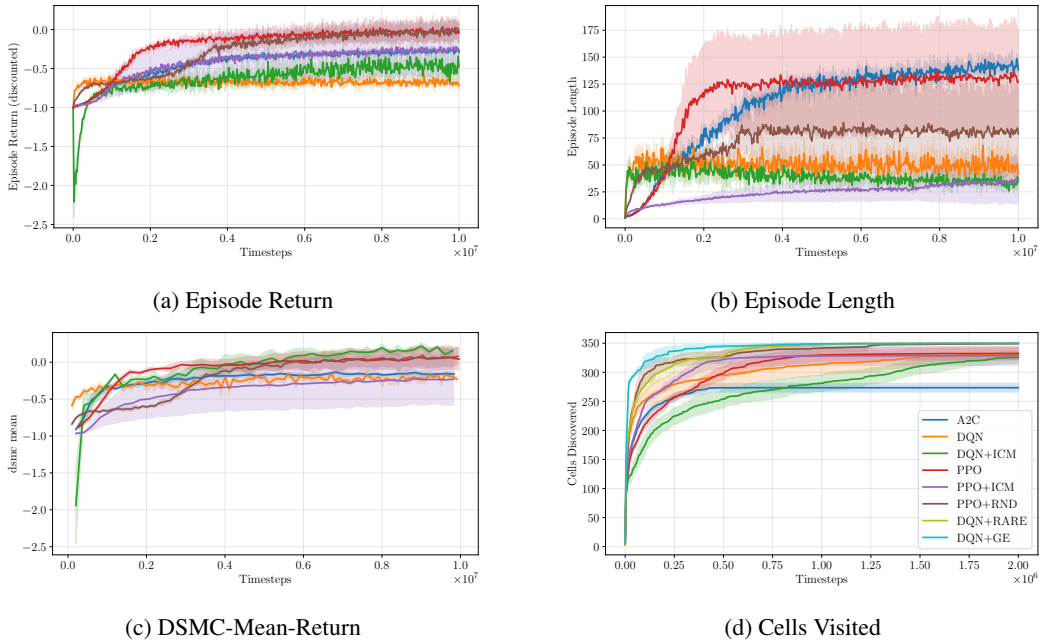
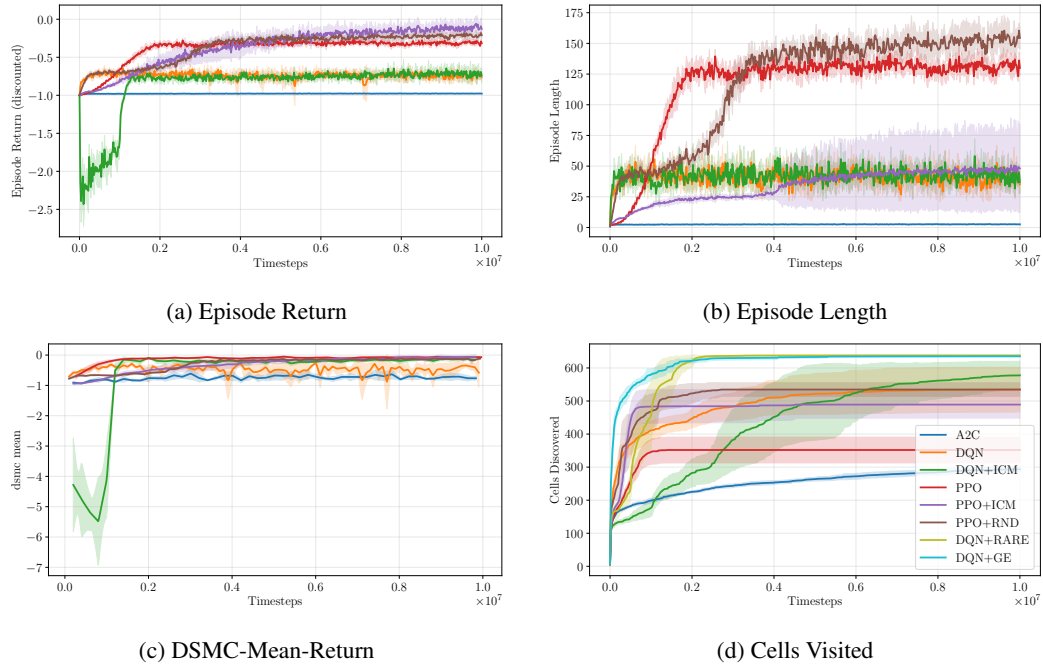
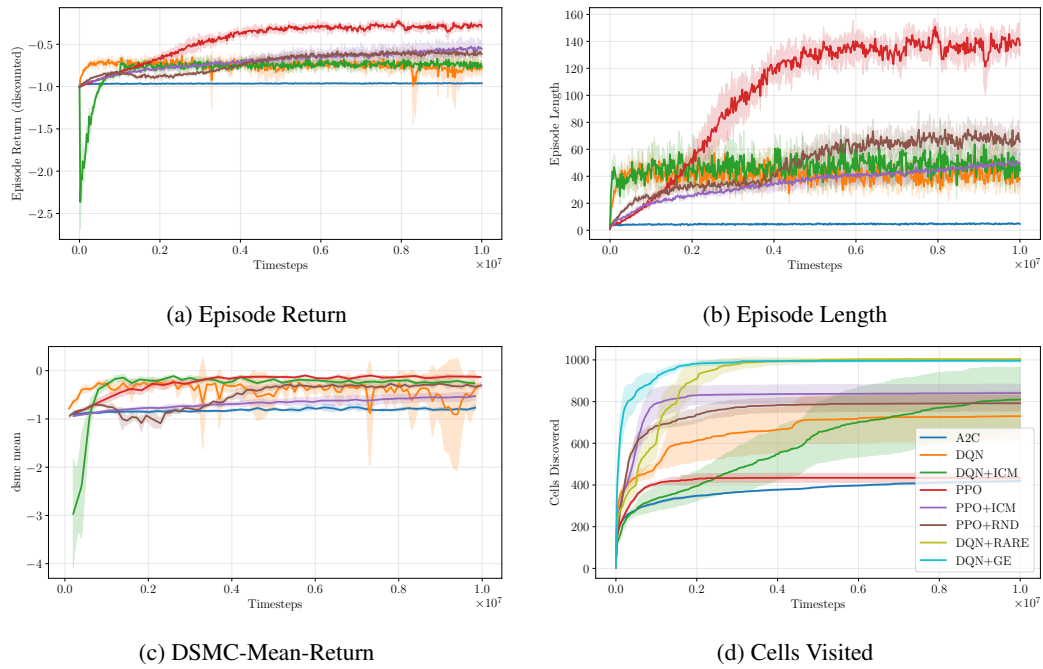


Figure 8: Training curves for the **exploration** scenario on 3×3 maps.

Figure 9: Training curves for the **exploration** scenario on 4×4 maps.Figure 10: Training curves for the **exploration** scenario on 5×5 maps.

D.2 Safety

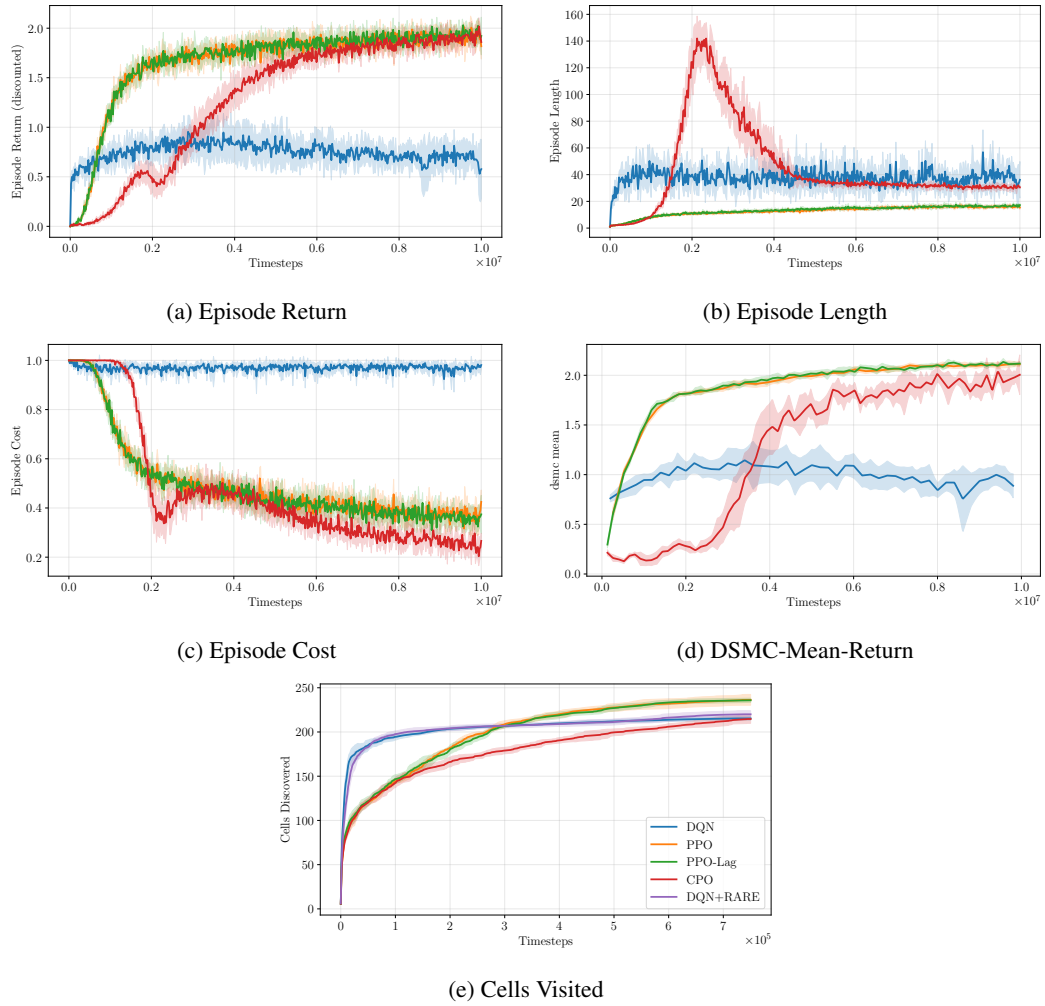
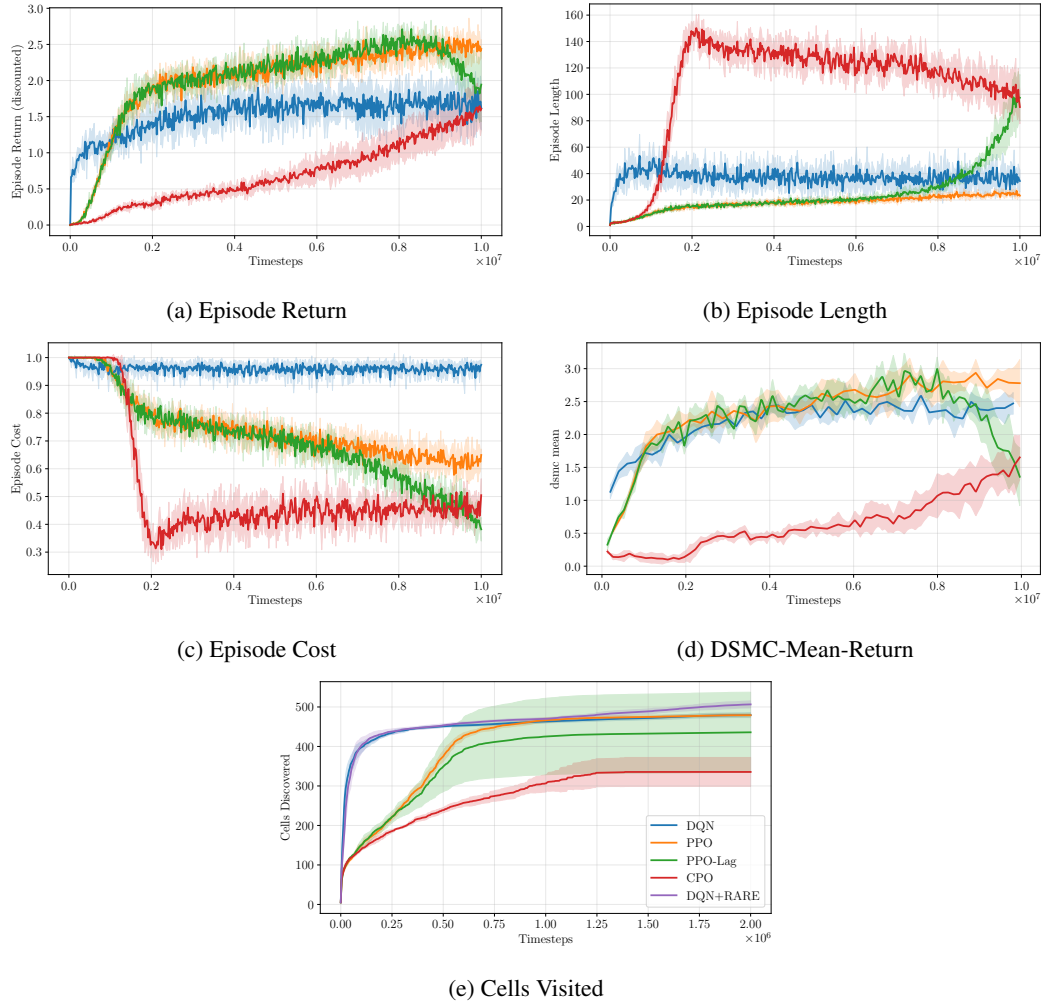


Figure 11: Training curves for the **safety** scenario on 2×2 maps.

Figure 12: Training curves for the **safety** scenario on 3×3 maps.

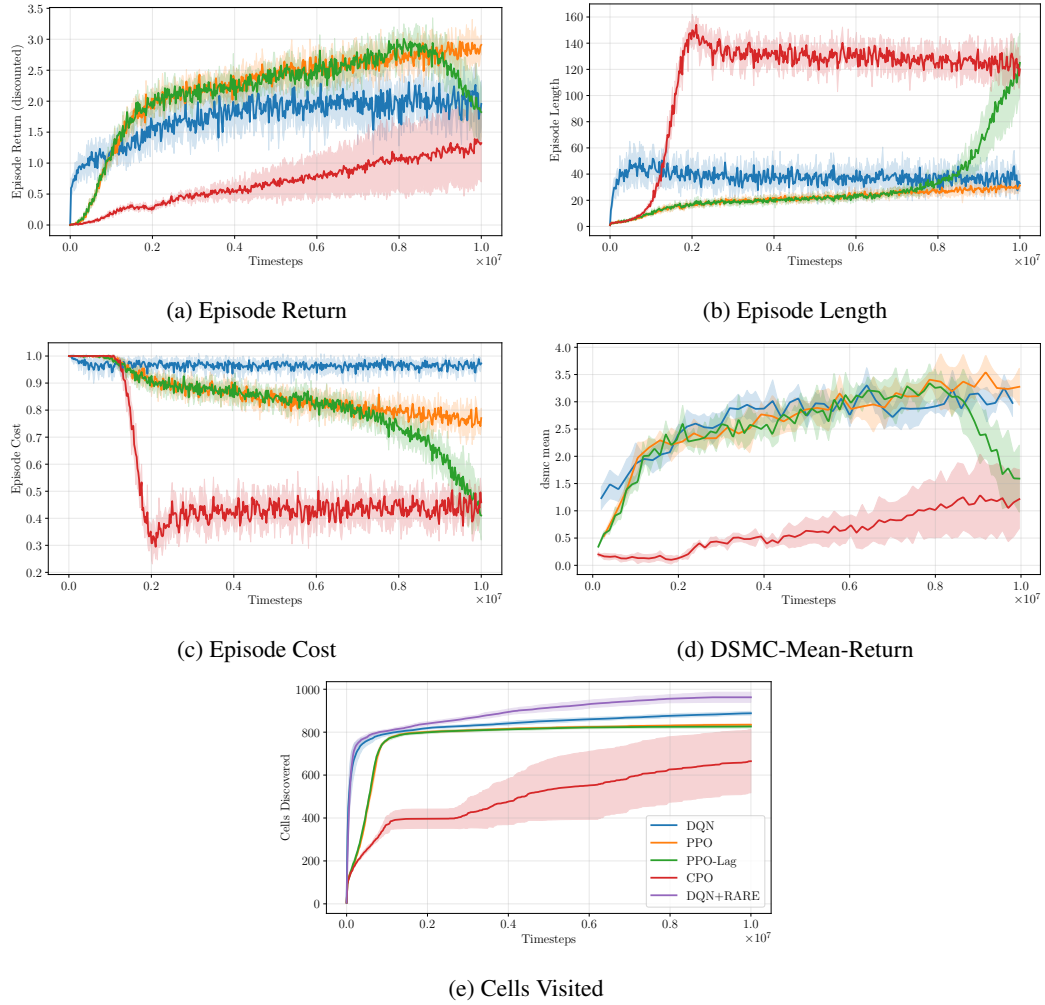


Figure 13: Training curves for the **safety** scenario on 4×4 maps.

D.3 Generalization

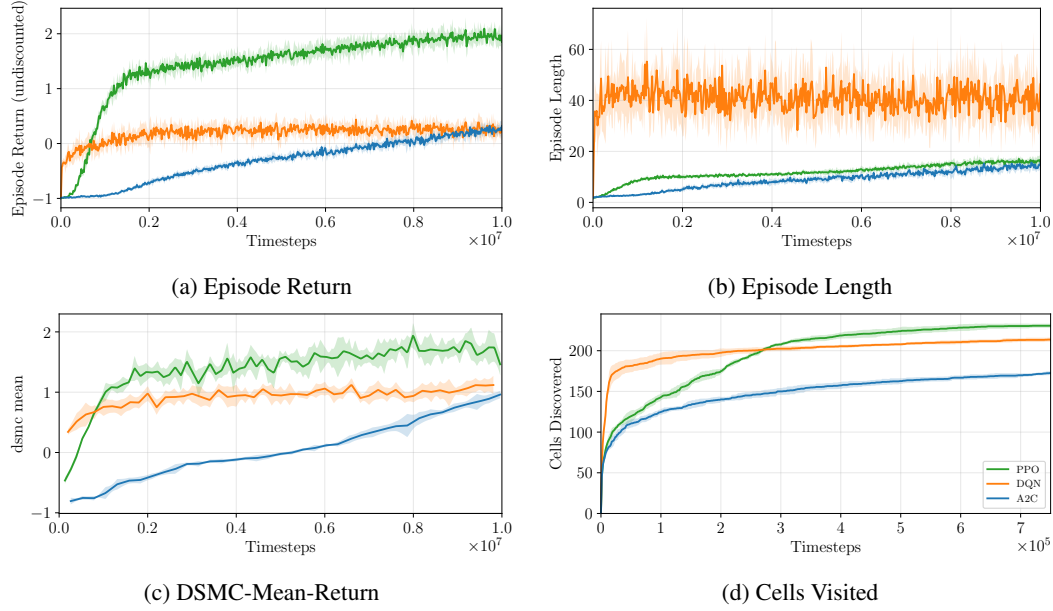


Figure 14: Training curves for the **generalization** scenario on 2×2 maps.

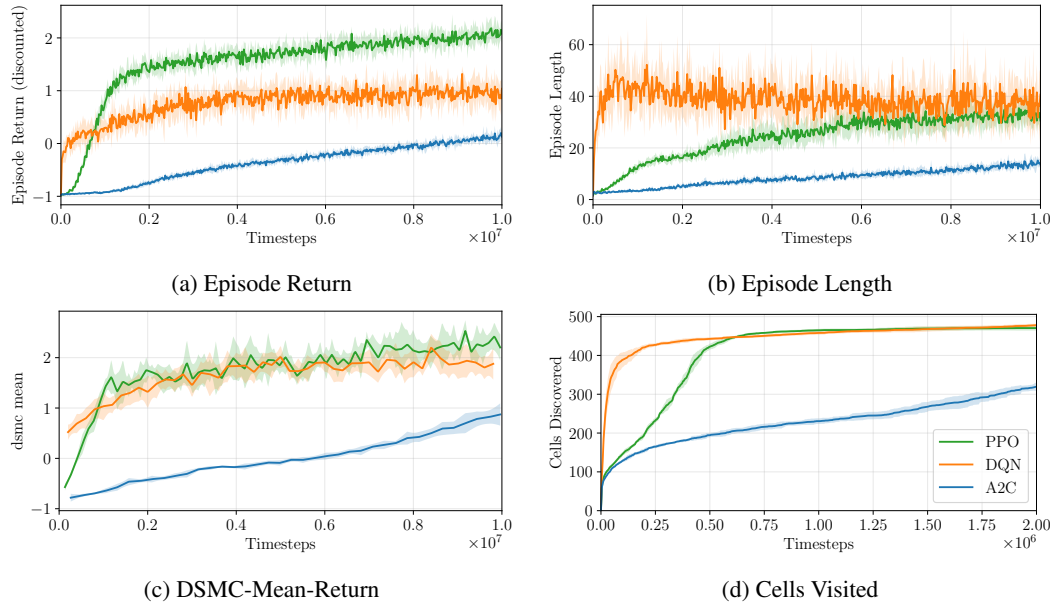
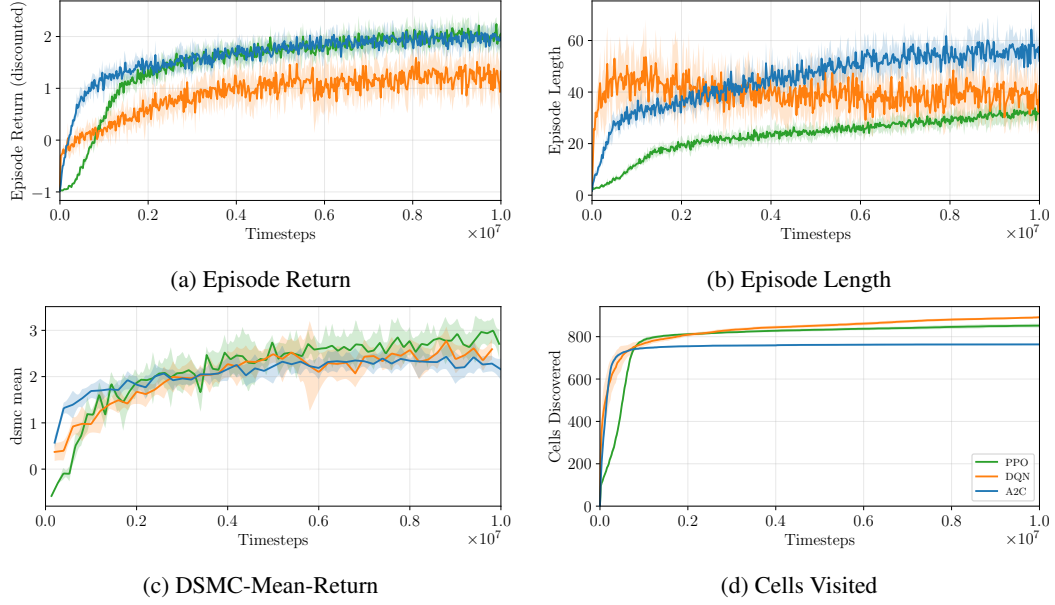


Figure 15: Training curves for the **generalization** scenario on 3×3 maps.

Figure 16: Training curves for the **generalization** scenario on 4×4 maps.

E Defining Custom Obstacles

```

1  @pgtg_obstacle("sand", act_probability=0.2)
2  def sand_behavior(env: PGTEEnv, local_ctx: StepContext) -> bool:
3      env.velocity = np.array([0, 0]) # Stuck in sand
4      local_ctx.remaining_substeps = [None] # Still validate the final destination tile
5      return False
6
7  @pgtg_obstacle("broken_road", act_probability=0.1, obstacle_weight=1.0, observe=True)
8  def broken_road_behavior(env: PGTEEnv, local_ctx: StepContext) -> bool:
9      env.flat_tire = True
10     return False
11
12  @pgtg_obstacle("custom_obstacle", act_probability=0.5, obstacle_weight=1.0)
13  def custom_obstacle_behavior(env: PGTEEnv, local_ctx: StepContext) -> bool:
14      # E.g., swap (x,y) components, and apply a small penalty
15      local_ctx.velocity_substep[[0, 1]] = local_ctx.velocity_substep[[1, 0]]
16      local_ctx.sum_penalty += 0.1
17      return False
18
19  # When creating the PGTEEnv, all you have to do is the following:
20  gym.make("pgtg-vX", # X = current PGTG version
21         obstacles = [
22             custom_obstacle_behavior(),
23             ice_behavior(observe=False),
24             broken_road_behavior(obstacle_weight=0.5),
25             sand_behavior(act_probability=0.75),
26             ], ...
27     )

```