Reinforcement Learning Algorithms for the Risk Sensitive Criterion

Velpula Chaithanya (22940)
Indian Institute of Science, Bengaluru
`chaithanyav@iisc.ac.in`

Guided by Prof. Shalabh Bhatnagar

Abstract

Conventional Reinforcement Learning (RL) algorithms primarily focus on maximizing the expected cumulative reward. However, such expectation-based optimization often fails to capture the underlying variability and risk inherent in real-world systems, where environmental noise and parametric uncertainties can significantly degrade policy performance. Even slight discrepancies between training and deployment conditions can result in poor generalization and unsafe behavior. To mitigate these limitations, we adopt a Risk-Sensitive Reinforcement Learning (RSRL) framework [1], which explicitly accounts for uncertainty by optimizing a risk-aware objective based on exponential utility functions. This formulation introduces regularization through an exponential criterion, effectively enhancing policy robustness and stability.

In particular, we propose a novel risk-sensitive Proximal Policy Optimization algorithm [2] and Soft Actor-Critic algorithm [2], designed to solve a multiplicative Bellman equation via stochastic approximation. The algorithm incorporates risk sensitivity directly into the policy update mechanism, enabling adaptive behavior under distributional shifts and stochastic perturbations. To empirically validate our approach, we simulate a self-driving car control scenario using the CARLA(Car Learning to Act) simulator—a high-fidelity autonomous driving environment. We modify the standard architecture [3] to support risk-sensitive policy updates within a custom traffic and weather-variant CARLA environment. The experimental outcomes, detailed in Section 5, demonstrate the superior robustness and generalization of risk-sensitive policies compared to their risk-neutral counterparts.

1 Introduction

Reinforcement Learning (RL) is widely used in stochastic decision-making systems to optimize long-term performance through trial-and-error interaction with an environment. Traditional RL methods typically adopt a *risk-neutral* objective, which focuses on maximizing the expected cumulative reward. While effective in many controlled settings, this approach often underperforms in real-world scenarios where environments are dynamic, noisy, or uncertain.

Risk-neutral RL evaluates performance based on averages, ignoring variability in outcomes. As a result, it fails to account for rare but impactful events—such as system perturbations, model mismatches, or environmental shifts—that are common in real-world applications. This limitation is particularly evident in standard *Actor-Critic* algorithms, where agents trained under fixed conditions may perform poorly when faced with even minor deviations at test time.

To overcome this fragility, *Risk-Sensitive Reinforcement Learning (RSRL)* has emerged as a robust alternative. Unlike its risk-neutral counterpart, RSRL incorporates uncertainty directly into the optimization objective. It accounts for both the expected return and the variability of outcomes, enabling the development of policies that are more resilient to fluctuations and disturbances.

Risk-sensitive methods achieve this by modifying the objective functions—often through constraints or alternative formulations such as exponential utility or variance penalties. These strategies promote policies that prioritize reliability and robustness over sheer reward maximization. Prior work by Moos et al. (2022), Osogami (2012), and Noorani & Baras (2021) has demonstrated that risk-aware objectives significantly enhance performance under uncertainty, making them better suited for real-world deployment.

2 RL Preliminaries and Policy Gradient Methods

This section introduces the foundational concepts of Reinforcement Learning (RL), including the Markov Decision Process (MDP) formulation, policy representations, and value functions. These preliminaries form the basis for modern RL algorithms, particularly **policy gradient methods** such as **Proximal Policy Optimization (PPO)** and **Soft Actor-Critic (SAC)**, which directly optimize the agent's behavior using gradient-based updates in continuous and high-dimensional action spaces.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a framework for sequential decision-making where an agent learns to act optimally by interacting with an environment. The mathematical foundation of RL is the **Markov Decision Process (MDP)**, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where:

- $\mathcal{S}$ is the set of all possible states,
- $\mathcal{A}$ is the set of all possible actions,
- $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function,
- $P(s'|s, a)$ is the transition probability model,
- $\gamma \in [0, 1)$ is the discount factor.

At each time step t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to a policy $\pi(a|s)$, receives a scalar reward $r_t = r(s_t, a_t)$, and transitions to a new state $s_{t+1} \sim P(\cdot|s_t, a_t)$. The reward signal is designed to encourage desirable behavior; for example, assigning low rewards for undesirable events (e.g., collisions) and high rewards for desirable events (e.g., lane following).

A **policy** $\pi(a|s)$ is a mapping from states to probability distributions over actions. The objective in RL is to find the **optimal policy** π^* that maximizes the expected **discounted return**:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

To evaluate and improve policies, RL relies on two key value functions:

- **State-Value Function**:

  $$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \middle| s_t = s \right]$$

  which represents the expected return when starting in state s and following policy π.

- **Action-Value Function**:

  $$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \middle| s_t = s, a_t = a \right]$$

  which represents the expected return when starting in state s, taking action a, and thereafter following policy π.

The optimal value functions are defined as:

$$V^*(s) = \max_\pi V^\pi(s), \quad Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

Once $Q^*(s,a)$ is learned, the **optimal policy** can be derived as:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a)$$

In practice, the agent estimates these optimal value functions through continuous interaction with the environment. Upon convergence, the agent behaves optimally with respect to the learned policy.

2.2 Policy Gradient Methods

Policy Gradient (PG) methods aim to directly learn a parameterized policy $\pi_\theta(a|s)$ that maximizes the expected cumulative reward. Unlike value-based methods, PG approaches are particularly suitable for continuous and high-dimensional action spaces.

The objective is:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right]$$

Using the Policy Gradient Theorem, the gradient of $J(\theta)$ can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \hat{A}_t\right]$$

where $\hat{A}_t$ is the **advantage function**:

$$\hat{A}_t = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

Baseline Subtraction and Variance Reduction

To reduce variance without introducing bias, a baseline function $b(s_t)$ is often subtracted:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot (R_t - b(s_t))\right]$$

A common choice is $b(s_t) = V^\pi(s_t)$, which gives the advantage formulation.

Entropy Regularization

To encourage exploration, an entropy term is added to the objective:

$$J_{\text{entropy}}(\theta) = J(\theta) + \lambda \mathbb{E}_{\pi_\theta}\left[\mathcal{H}(\pi_\theta(\cdot|s_t))\right]$$

where $\mathcal{H}(\pi) = -\sum_a \pi(a|s) \log \pi(a|s)$ and λ controls the exploration-exploitation balance.

Proximal Policy Optimization (PPO)

PPO [**schulman2017proximal**] improves policy stability by using a clipped surrogate objective. Define the **probability ratio**:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

The clipped objective is:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

This prevents large updates by constraining the ratio $r_t(\theta)$ within a trust region $[1-\epsilon, 1+\epsilon]$, where ϵ is typically 0.1–0.3. The final PPO objective includes an entropy bonus:

$$L^{\text{PPO-full}}(\theta) = L^{\text{PPO}}(\theta) + \lambda \mathbb{E}_t\left[\mathcal{H}(\pi_\theta(\cdot|s_t))\right]$$

Soft Actor-Critic (SAC)

SAC [**haarnoja2018soft**] is an off-policy algorithm that optimizes a stochastic policy in the **maximum entropy** framework. The objective is to maximize both reward and entropy:

$$J(\pi_\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[Q(s_t,a_t) - \alpha\log\pi_\theta(a_t|s_t)\right]$$

Here:

- $\alpha > 0$ is the temperature parameter, balancing reward and entropy,
- $\mathcal{D}$ is a replay buffer of sampled transitions.

The policy is updated by minimizing:

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim\mathcal{D},a_t\sim\pi_\theta}\left[\alpha\log\pi_\theta(a_t|s_t) - Q_\phi(s_t,a_t)\right]$$

The Q-function is updated by minimizing the soft Bellman residual:

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim\mathcal{D}}\left[\left(Q_\phi(s_t,a_t) - y_t\right)^2\right]$$

with target:

$$y_t = r_t + \gamma\mathbb{E}_{a_{t+1}\sim\pi_\theta}\left[Q_{\bar{\phi}}(s_{t+1},a_{t+1}) - \alpha\log\pi_\theta(a_{t+1}|s_{t+1})\right]$$

To stabilize learning, SAC uses:

- Two Q-networks and a target network,
- Polyak averaging for target updates,
- Automatic entropy tuning by minimizing:

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_\theta}\left[-\alpha\left(\log\pi_\theta(a_t|s_t) + \mathcal{H}_{\text{target}}\right)\right]$$

Summary

Policy Gradient methods optimize policies through gradient ascent. PPO ensures reliable learning via clipped updates, while SAC encourages exploration and robustness with entropy regularization and off-policy learning.

Table 1: Comparison of PPO and SAC

Aspect	**PPO**	**SAC**
Policy Type	Stochastic	Stochastic
Learning Type	On-policy	Off-policy
Exploration Mechanism	Entropy bonus (optional)	Max entropy (mandatory)
Sample Efficiency	Lower	Higher
Stability	High	High
Action Space	Continuous/Discrete	Typically continuous

3 Risk-Sensitive Reinforcement Learning and Simulator

We provide a brief overview of Risk-Sensitive Reinforcement Learning (RSRL), an extension of conventional reinforcement learning that incorporates robustness and uncertainty-awareness into the learning objective. This is especially critical for real-world deployment scenarios, where unpredictable environmental perturbations and model uncertainties can significantly degrade the performance of risk-neutral policies.

To evaluate the effectiveness of RSRL methods, we employ the CARLA simulator, a high-fidelity environment for autonomous driving research. This section introduces the CARLA-based experimental setup, detailing the environment configuration, the design of state and action spaces, and the custom reward function tailored to reflect risk-sensitive criteria.

3.1 Risk-Sensitive Reinforcement Learning

Risk-Sensitive Reinforcement Learning (RSRL) [1] enhances traditional RL by accounting for the variability in returns, allowing policies to be optimized not just for reward maximization but also for robustness under uncertainty.

In a standard Markov Decision Process (MDP), the objective is to maximize expected cumulative reward:

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta} [R(\tau)] = \mathbb{E}_{\tau \sim \rho_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right],$$

where $\tau = (s_0, a_0, \ldots, s_T)$ is a trajectory, $R(\tau)$ is the total reward, and ρ_θ is the trajectory distribution under policy π_θ.

RSRL introduces an exponential utility objective:

$$J_\beta(\theta) = \frac{1}{\beta} \log \mathbb{E}_{\tau \sim \rho_\theta} \left[e^{\beta R(\tau)} \right],$$

where $\beta \in \mathbb{R}$ is the **risk sensitivity parameter**:

- $\beta > 0$: risk-seeking (prefers high-variance rewards),
- $\beta < 0$: risk-averse (prefers low-variance, stable rewards).

Using Taylor expansion, this can be approximated as:

$$J_\beta(\theta) \approx \mathbb{E}[R(\tau)] + \frac{\beta}{2} \text{Var}[R(\tau)] + \mathcal{O}(\beta^2),$$

where $\text{Var}[R(\tau)] = \mathbb{E}[(R(\tau) - \mathbb{E}[R(\tau)])^2]$.

The second term, $\frac{\beta}{2}\text{Var}[R(\tau)]$, modulates the objective based on reward variance: If $\beta < 0$, this term penalizes high-variance trajectories, guiding the policy toward more consistent and predictable outcomes.

Environmental perturbations are modeled as:

$$P'(s'|s,a) = P(s'|s,a) + \Delta P(s'|s,a), \quad r'(s,a) = r(s,a) + \Delta r(s,a),$$

leading to changes in trajectory distribution and expected return. The effect on the expected reward is approximated as:

$$\Delta \mathbb{E}[R(\tau)] \propto \sqrt{\text{Var}[R(\tau)]} \cdot \|\Delta\|,$$

where $\|\Delta\|$ denotes the magnitude of environmental perturbations.

This shows that:

- **Higher variance** increases the sensitivity of the expected return to perturbations.
- **Risk-averse optimization** (i.e., $\beta < 0$) naturally reduces this sensitivity by discouraging high-variance trajectories.

Hence, minimizing $\text{Var}[R(\tau)]$ through a negative β enhances the **robustness** of the policy:

$$\boxed{\text{Robustness} \uparrow \quad \text{as} \quad \beta \downarrow < 0 \quad \text{and} \quad \text{Var}[R(\tau)] \downarrow}$$

This trade-off between reward and robustness makes RSRL particularly effective in uncertain, dynamic, or adversarial environments.

3.2 CARLA (Car Learning to Act) Simulator

CARLA is an open-source, high-fidelity simulator developed for autonomous driving research by the Computer Vision Center (CVC) at the Universitat Autònoma de Barcelona. It provides photorealistic rendering, realistic physics, and a flexible API for vehicle and sensor control.

Key features of CARLA include:

- Realistic 3D urban environments and vehicle dynamics,
- A rich suite of configurable sensors (camera, LiDAR, GPS, IMU),
- Multi-agent simulation support,
- Full Python API and ROS compatibility.

4 Related Work

This section presents the simulation environment and learning framework used to train autonomous driving agents in CARLA. We describe how sensor inputs are captured and preprocessed, how spatial and dynamic features are fused into a unified state representation, and how policy and value functions are trained using deep reinforcement learning algorithms such as Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC).

4.1 Image Capture and Preprocessing in CARLA

In the CARLA simulator, an RGB camera sensor captures forward-facing images at a resolution of $800 \times 600 \times 3$ pixels. The preprocessing steps include:

- Conversion of RGB to grayscale,
- Resizing to 128×128,
- Normalization to $[-1, 1]$ using:

$$\text{normalized\_pixel} = \frac{\text{pixel}}{255.0} - 0.5$$

This preprocessing reduces computational overhead and ensures consistent input scaling.

4.2 State Representation Using ResNet-50 and Dynamic Features

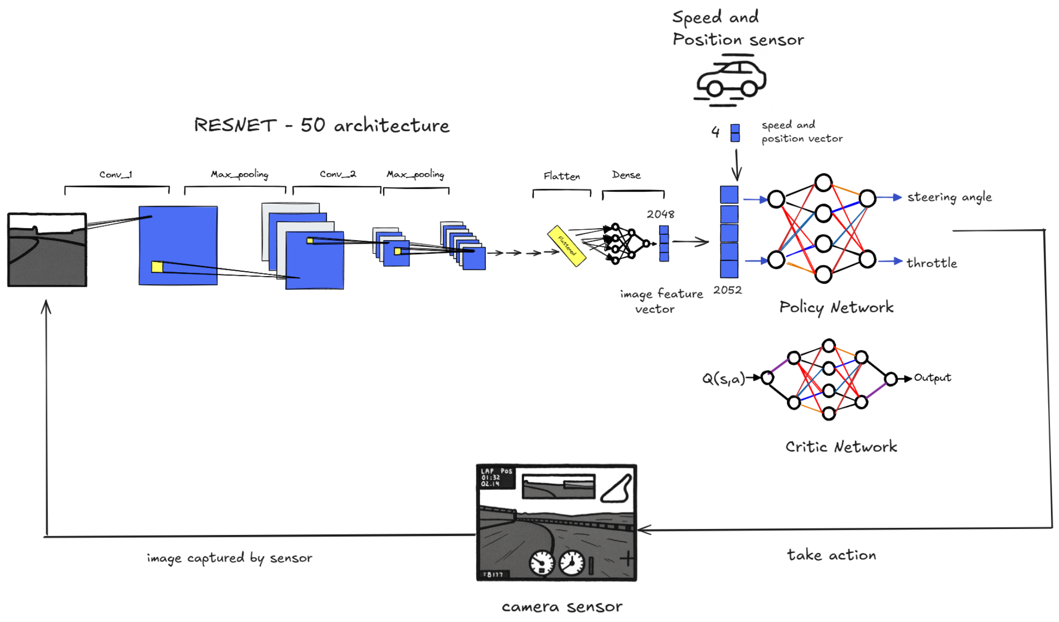

Figure 1: State representation and control pipeline: visual features extracted from a camera image using ResNet-50 are concatenated with dynamic features (speed, throttle, relative position) to form a 2052-dimensional state vector. This state is then processed by the policy and critic networks to generate control actions and value estimates.

The final state representation used for control comprises:

- **Visual Features:** Extracted from the camera image using a pre-trained ResNet-50 network, yielding a 2048-dimensional embedding.
- **Dynamic Features:** A 4-dimensional vector encoding speed, throttle, and relative positional information.

These are concatenated to form a 2052-dimensional state vector, which is fed into both the policy and critic networks.

4.3 Simulation Environment

At each discrete time step, the agent receives a *state* from its sensors, selects an *action*, and receives a *reward*. The environment transitions to the next state based on the underlying simulation dynamics.

State Representation. The state s_t includes:

- Frontal camera image,
- Positional coordinates of the vehicle,
- Engine speed.

Action Space. The agent controls the vehicle using:

$$a_t = [\text{steering},\ \text{throttle}] \in [-1, 1] \times [0, 1]$$

- Steering $\in [-1, 1]$: full-left to full-right,
- Throttle $\in [0, 1]$: no acceleration to full throttle.

Reward Function

The reward at time t is computed as:

$$R_t = \lambda_1 R_{\text{safety}} + \lambda_2 R_{\text{progress}} + \lambda_3 R_{\text{comfort}} + \lambda_4 R_{\text{rules}}$$

with weighting coefficients:

$$\lambda_1 = 1.5, \quad \lambda_2 = 1.0, \quad \lambda_3 = 0.8, \quad \lambda_4 = 1.2$$

Each component is defined as follows:

$$R_{\text{safety}} = \begin{cases} -300 \cdot (1 + \min(\frac{v}{50}, 1)) & \text{if collision occurs} \\ -\frac{5}{\text{TTC}} & \text{if TTC} < 2 \\ +5 & \text{otherwise} \end{cases}$$

$$R_{\text{progress}} = \begin{cases} 2.0 \cdot \exp(-\frac{d}{10}) & \text{if } d \leq 10 \\ -0.5 & \text{otherwise} \end{cases} + 5 \cdot \max\left(0, \frac{v}{40}\right)$$

$$R_{\text{comfort}} = -|\Delta\text{throttle}| - 0.1 \cdot |\text{steering}|$$

$$R_{\text{rules}} = -5 \cdot \nVdash_{v>50} - 10 \cdot \nVdash_{\text{lane violation}} - 7 \cdot \nVdash_{\text{headway violation}}$$

Risk-Sensitive Reward Adjustment

To enhance robustness and account for reward variance, we apply an exponential utility-based adjustment to the cumulative reward:

$$\widetilde{R}_t = -\frac{1}{\beta} \log\left(\mathbb{E}\left[\exp(-\beta R_t)\right]\right)$$

This is approximated using a finite reward history of N samples:

$$\widetilde{R}_t \approx -\frac{1}{\beta} \log\left(\frac{1}{N}\sum_{i=1}^{N} \exp\left(-\beta \frac{R_i}{s}\right)\right)$$

where β is the risk sensitivity parameter, and s is a scaling factor (typically set to 5000 in experiments).

Termination Conditions

An episode terminates if any of the following conditions are met:

- A collision is detected,
- The vehicle deviates significantly from the target path ($d > 10$),
- The episode exceeds a predefined maximum number of steps (5000 steps).

4.4 Learning Algorithms: PPO and SAC

Both PPO and SAC are employed to train the autonomous driving agent:

- **PPO** stabilizes on-policy learning using clipped objective functions and advantage estimates.
- **SAC** performs off-policy entropy-regularized learning to encourage exploration and robustness.

These algorithms are implemented using the Stable-Baselines3 framework and trained in an end-to-end fashion, with ResNet feature extraction included in the forward pass.

4.5 Evaluation Under Weather Perturbations

To evaluate generalization, the trained policies are tested in altered environments featuring different weather conditions (e.g., heavy rain). The weather is set via the CARLA weather API using:

$$\text{precipitation} = \text{rain intensity} \in [0, 100]$$

This allows the assessment of policy robustness and sim-to-real generalization.

4.6 Risk-Sensitive Learning Algorithms

The following algorithms summarize the training procedures for R-PPO and R-SAC:

Algorithm 1 Risk-sensitive Proximal Policy Optimization (R-PPO)

```
Input: stochastic policy $\pi(a|s;\theta)$, value function $V(s;\phi)$
Parameters: learning rate $\alpha$, risk sensitivity $\beta$, clip threshold $\epsilon$, epochs $K$
 1: Initialize policy parameters $\theta_0$, value parameters $\phi_0$
 2: for each iteration do
 3:     Collect trajectories $\{\tau_i\}$ using current policy $\pi(\cdot|\cdot;\theta)$
 4:     for each trajectory $\tau$ do
 5:         for each timestep $t$ in $\tau$ do
 6:             Compute exponential return:
                    $$\hat{R}_t^\beta \leftarrow \frac{1}{\beta}\log\left(\sum_{k=t}^{T} e^{\beta r_k}\right)$$
 7:             Compute risk-sensitive advantage: $\hat{A}_t^\beta \leftarrow \hat{R}_t^\beta - V(s_t;\phi)$
 8:             Compute ratio: $r_t(\theta) \leftarrow \frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}$
 9:             PPO surrogate loss:
                    $$L_t^\beta(\theta) \leftarrow \min\left(r_t(\theta)\cdot\hat{A}_t^\beta,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\cdot\hat{A}_t^\beta\right)$$
10:         end for
11:     end for
12:     Update policy: $\theta \leftarrow \theta + \alpha\nabla_\theta \sum_t L_t^\beta(\theta)$
13:     Update value function: $\phi \leftarrow \phi - \alpha\nabla_\phi \sum_t (V(s_t;\phi) - \hat{R}_t^\beta)^2$
14: end for=0
```

Algorithm 2 Risk-sensitive Soft Actor-Critic (R-SAC)

```
Input: policy $\pi(a|s;\theta)$, Q-networks $Q_1$, $Q_2$ with targets $Q_1'$, $Q_2'$
Parameters: learning rates $\eta, \zeta$, temperature $\alpha$, risk parameter $\beta$
 1: Initialize policy parameters $\theta$, Q-network parameters $\phi_1, \phi_2$, target networks $\phi_1', \phi_2'$
 2: Initialize replay buffer $\mathcal{D}$
 3: for each iteration do
 4:     Collect experience $(s, a, r, s')$ using $\pi(a|s;\theta)$, add to $\mathcal{D}$
 5:     Sample mini-batch $\{(s_i, a_i, r_i, s_i')\}$ from $\mathcal{D}$
 6:     for each sample do
 7:         Compute exponential risk-sensitive target:
                $$y_i^\beta = \frac{1}{\beta}\log\left(e^{\beta r_i} + \gamma\mathbb{E}_{a'\sim\pi}\left[e^{\beta(Q'_{\min}(s_i', a') - \alpha\log\pi(a'|s_i'))}\right]\right)$$
 8:         Update critics:
                $$\phi_j \leftarrow \phi_j - \eta\nabla_{\phi_j}\left(Q_{\phi_j}(s_i, a_i) - y_i^\beta\right)^2,\ j = 1, 2$$
 9:         Update policy:
                $$\theta \leftarrow \theta + \zeta\nabla_\theta\left(\alpha\log\pi(a_i|s_i;\theta) - Q_{\min}(s_i, a_i)\right)$$
10:     end for
11:     Update target networks: $\phi_j' \leftarrow \tau\phi_j + (1-\tau)\phi_j'$
12: end for=0
```

5 Results

We report and analyze the performance of agents trained using Risk-sensitive PPO (R-PPO) and Risk-sensitive SAC (R-SAC) for different risk aversion levels $\beta \in \{0, 1, 5, 10\}$. Results are grouped into training behavior, generalization performance under weather perturbations, and metric-based evaluation using mean reward, CVaR, and entropic risk.

5.1 Training Curves for R-PPO and R-SAC

Figure **??** and Figure **??** show the learning curves for R-PPO and R-SAC across four β values. As risk sensitivity increases, both algorithms exhibit smoother and more cautious learning behavior, with reduced reward variance and fewer outliers.

Training Reward Curves – R-PPO

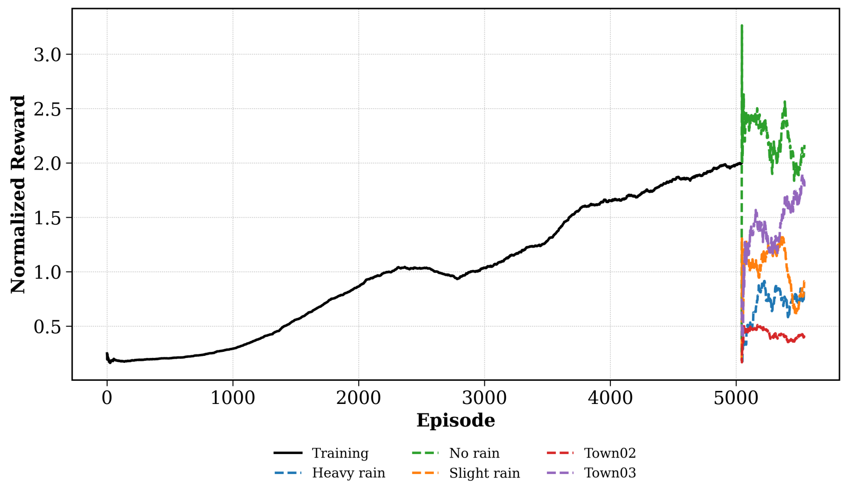

Figure 2: Training reward trajectory for R-PPO with $\beta = 0$.

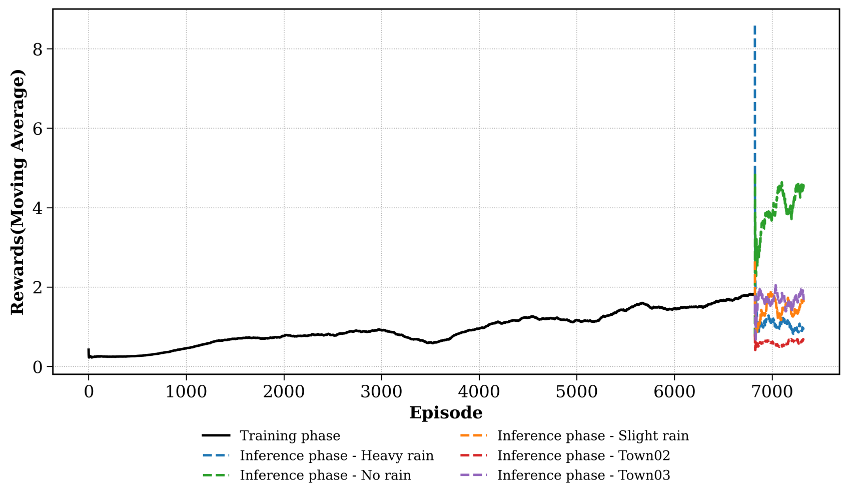

Figure 3: Training reward trajectory for R-PPO with $\beta = 5$.

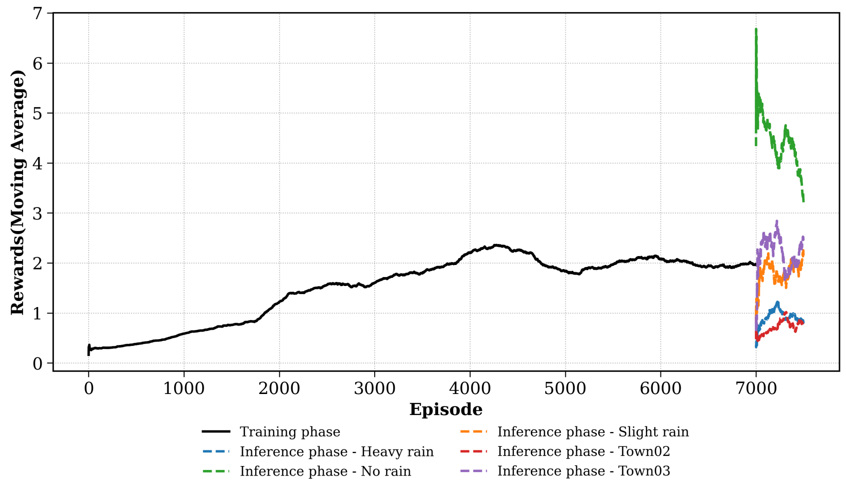

Figure 4: Training reward trajectory for R-PPO with $\beta = 10$.

Training Reward Curves – R-SAC (1+2 Grid)

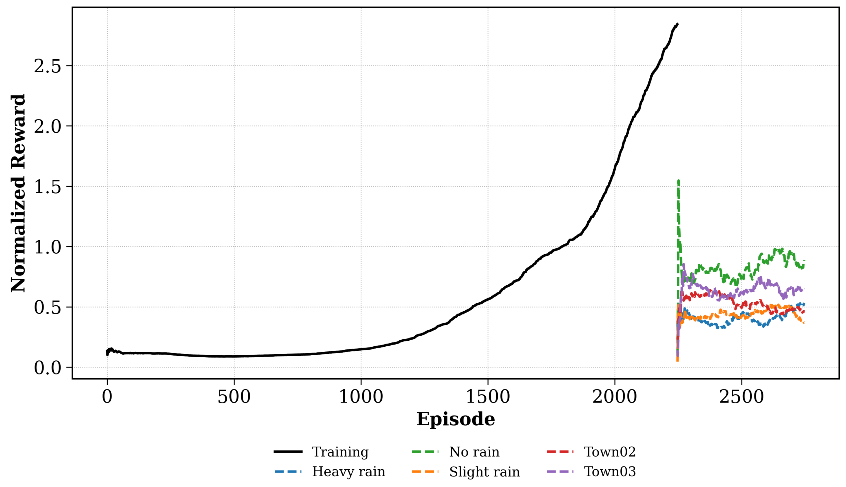

Figure 5: Training reward trajectory for R-PPO with $\beta = 0$.

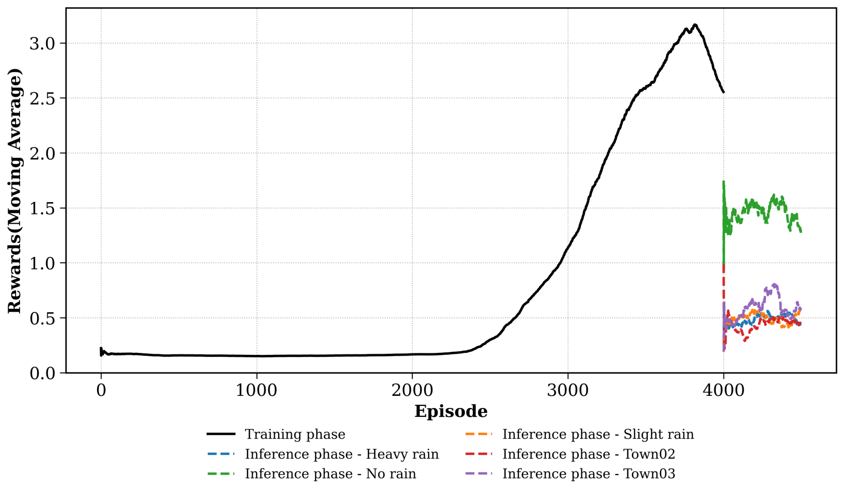

Figure 6: Training reward trajectory for R-PPO with $\beta = 5$.

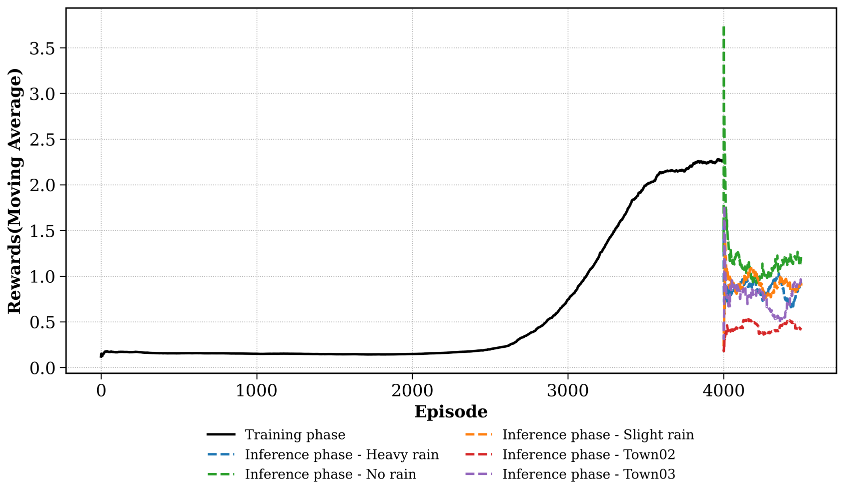

Figure 7: Training reward trajectory for R-PPO with $\beta = 10$.

5.2 Risk-Aware Performance Metrics

We computed the following metrics using episode-wise rewards for each perturbation setting:

- **Mean Reward:** $\mathbb{E}[R] = \frac{1}{N}\sum_{i=1}^{N} R_i$
- **CVaR (at $\alpha = 0.99$):** average of the bottom 1% of rewards
- **Entropic Risk:** $\rho_\beta(R) = -\frac{1}{\beta}\log \mathbb{E}[\exp(-\beta R)]$

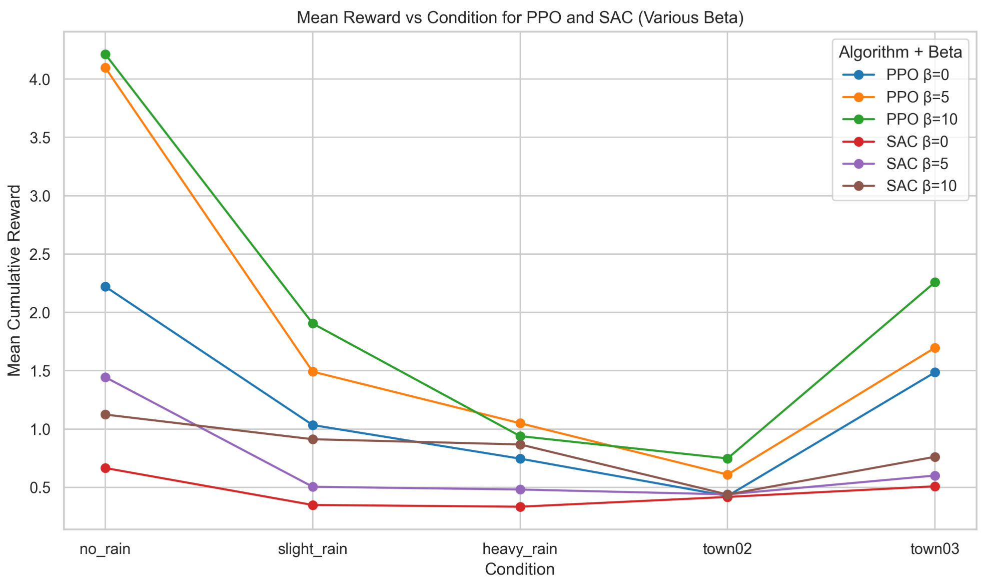

Figure 8: Mean reward vs perturbation for different β.

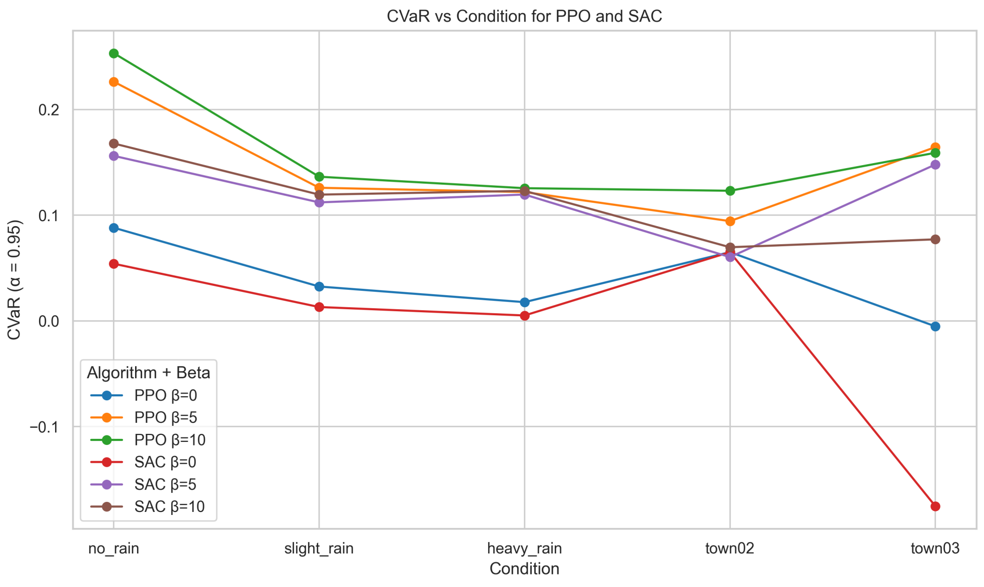

Figure 9: CVaR ($\alpha = 0.99$) vs perturbation for different β.

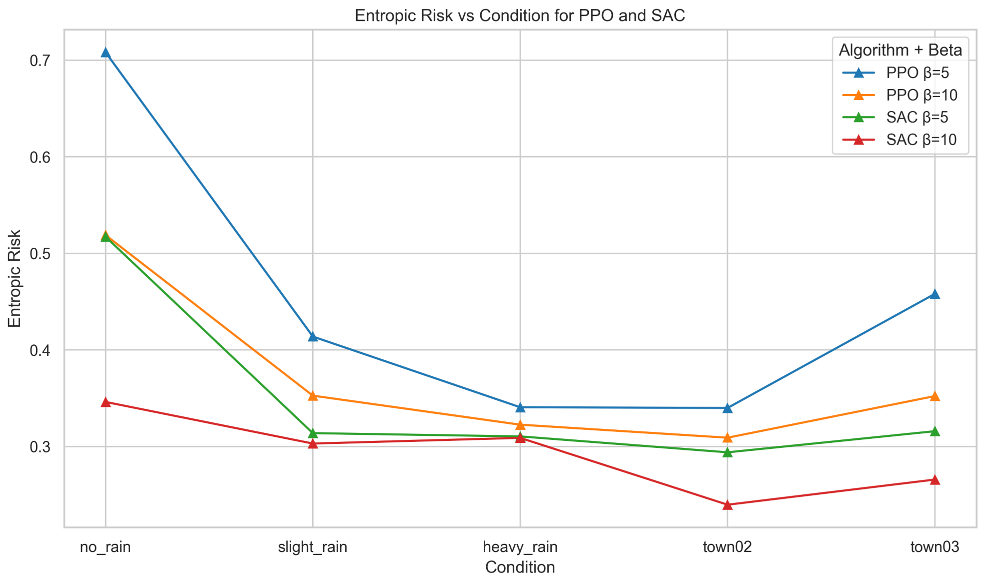

Figure 10: entropic$_risk(\alpha = 0.99)$ vs perturbation for different β.

5.3 Summary of Observations

- Higher β values result in more stable training, conservative policies, and better generalization.
- Risk-sensitive agents outperform risk-neutral ones in CVaR and entropic risk, especially under high perturbation.
- R-SAC demonstrates stronger resilience than R-PPO across all metrics, leveraging its entropy-based learning.
- Unified risk-sensitive metrics provide better insight into policy safety than mean reward alone.

These results highlight the effectiveness of entropic utility-based reward shaping in reinforcement learning for safe and robust autonomous driving.

6 Future Work

Future work will focus on applying risk-sensitive criteria to various reinforcement learning (RL) algorithms and designing new environments to evaluate their effectiveness. The main directions are as below:

- Modify RL algorithms such as Deep Q-Network (DQN) and Actor-Critic to incorporate risk-sensitive objectives, such as variance or other risk measures.
- Develop environments with altered reward functions, introducing noise, variability, or adversarial conditions to assess policy performance under uncertainty.

These efforts aim to make reinforcement learning algorithms more robust and adaptable to uncertain and complex environments.

References

[1] Erfaun Noorani, Christos Mavridis, and John Baras. *Risk-Sensitive Reinforcement Learning with Exponential Criteria*. 2024. arXiv: 2212.09010 [eess.SY]. URL: https://arxiv.org/abs/2212.09010.

[2] Vijay Konda and Vijaymohan Gao. “Actor-critic algorithms”. In: (Jan. 2000).

[3] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG]. URL: https://arxiv.org/abs/1312.5602.