



SPoRt – Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL

Jacques Cloete, Nikolaus Vertovec, Alessandro Abate

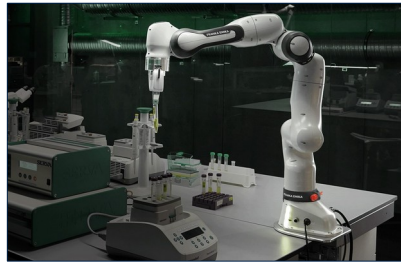
University of Oxford

Hello, my name is Jacques, I'm from the University of Oxford Control and Verification group, and I'll be talking about our paper, "Safe Policy Ratio" (or SPoRt).

- To apply RL to **safety-critical applications**, we ought to provide **safety guarantees** during both policy training and deployment



Credit: Waymo



Credit: Franka Robotics



Credit: Amazon

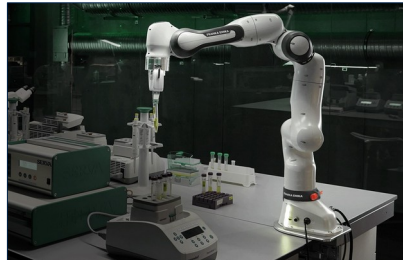
To apply reinforcement learning to safety-critical applications, we ought to provide safety guarantees during both training and deployment.

Safety during training is particularly important if we're training in real-world environments, which is often the case in robotics, for example.

- In many cases we would also like to **adapt an existing safe policy** to maximize reward **while maintaining these safety guarantees**



Credit: Waymo



Credit: Franka Robotics






Credit: Amazon

We may also want to adapt an existing “safe” policy to maximize some new reward while maintaining these safety guarantees.

For example, we may want our delivery drone to be faster or more energy efficient but still safely reach its destination.

Introducing Safe Policy Ratio (SPoRt)



-  **Model-free** RL approach (for episodic tasks of bounded length)
-  **Adapts** an existing ‘safe’ policy to **maximize task-specific reward**
-  **Maintains a bound on safety violation**, enforced during training and known prior to rollout

SPoRt - Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL (Cloete et. al.)

4

In response to this, we present Safe Policy Ratio (or SPoRt);

a model-free RL approach that adapts an existing ‘safe’ policy to maximize task-specific reward while maintaining a bound on safety violation, enforced during training and known prior to rollout.

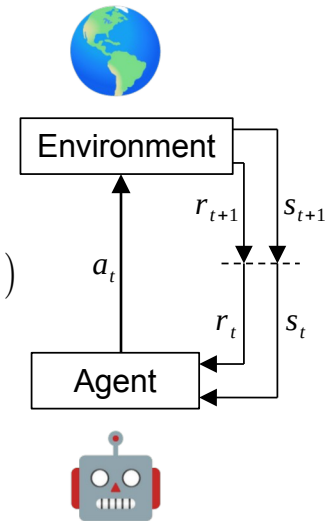
In other words, SPoRt enables users to trade off safety guarantees for task-specific performance, while maintaining a prior bound on safety.

I’ll take you through the approach now.

Problem Setup

🌍 MDP $\langle S, A, p, \mu, r_{task} \rangle$, safety constraints φ

- Continuous states $s \in S$ and actions $a \in A$
- Initial state distribution $\mu(s) \in \Delta(S)$
- State transition distribution $p(s'|a, s) : S \times A \rightarrow \Delta(S)$
- Task-specific reward $r_{task}(s, a) : S \times A \rightarrow \mathbb{R}$
- Episode length T



🤖 Policy $\pi(a|s) : S \rightarrow \Delta(A)$

We begin with an overview of our model-free RL setup.

The agent interacts with an unknown environment modelled as a Markov Decision Process, with continuous states and actions, and a task-specific reward function.

- Define safety in terms of **temporal property** φ
- Use **Linear Temporal Logic** (LTL)
 - Great for defining temporally-extended properties
 - e.g. $\varphi = (\neg \text{'hazard'}) \ U \ \text{'goal'}$ (*reach-avoid property*)
 - Binary metric for safety violation
- Train policies for LTL satisfaction using e.g. LCRL (*Hasanbeig et al*)

Now, let's define safety as a temporal property, in particular, a Linear Temporal Logic formula.

LTL is great for specifying constraints in safety-critical systems because it allows for the formal expression of temporally-extended properties in a way that is precise and verifiable.

A good example is the reach-avoid property, or “avoid the hazard until you’ve reached the goal”.

Note that we can train RL agents to satisfy LTL properties, using methods like LCRL.

- Distinct from (task-specific) reward!
 - φ represents what we require the agent to do (reach goal/avoid hazard)
 - r_{task} represents a performance metric to improve (speed/energy)

Note that the safety property is distinct from the task-specific reward.

The property represents what we require the agent to do (such that failing to do this counts as safety violation), e.g., reach-avoid.

Meanwhile, the reward represents a performance metric to improve, such as time taken or power usage.

With this in mind, SPoRt is best suited to improving performance of something the agent can already do, rather than getting it to do something entirely new.

- Evaluate φ on finite **trajectory** $\tau_\pi = (s_0, \dots, s_T)$ under policy π
 - $\tau_\pi \not\models \varphi$ means safety violation, $\tau_\pi \models \varphi$ means satisfaction

To determine safety violation, we evaluate the property on a finite trajectory of states, equal to the episode length, generated using our policy.

- Existing **base policy** π_{base} that we *think* is safe (*i.e.* $\tau_{\pi} \models \varphi$)
 - How to actually **certify** safety in a **model-free** setup?
- Use **PAC learning** to compute a bound on violation probability

$$P(\tau_{\pi_{base}} \not\models \varphi) \leq \epsilon_{base}$$

Bound on violation probability for base policy

- E.g. **scenario approach** (Campi & Garatti)

Now, imagine we're given an existing base policy that we think is safe, perhaps trained in simulation using LCRL.

How do we actually certify safety in a model-free setup?

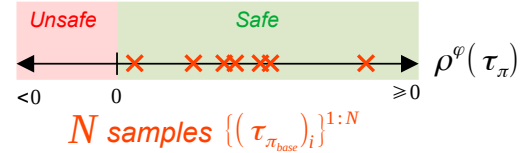
The solution is to use some kind of PAC learning to bound the probability of safety violation with high confidence.

There are different methods one could use, but for our paper we used the scenario approach.

- Robustness metric $\rho^\varphi : S^T \rightarrow \mathbb{R}$

- $\tau_\pi \not\models \varphi \Leftrightarrow \rho^\varphi(\tau_\pi) < 0$
- $\tau_\pi \models \varphi \Leftrightarrow \rho^\varphi(\tau_\pi) \geq 0$

How likely that a new sample $\tau_{\pi_{base}}$ is unsafe?



If $\rho^\varphi(\tau_{\pi_{base}}) \geq 0$ for N samples $\{(\tau_{\pi_{base}})_i\}^{1:N}$,
 then $P(\rho^\varphi(\tau_{\pi_{base}}) < 0) \leq \epsilon_{base}$
 with confidence $1 - \beta$, where $\beta = (1 - \epsilon_{base})^N$

I'm happy to go into more details about how we used scenario approach in the discussion.

(For any property there exists a real-valued robustness metric that is negative on property violation and non-negative on satisfaction.

Scenario approach tells us that, if we collect N trajectory rollouts under the base policy with all non-negative metric values, then we can bound the probability that a new trajectory rollout has negative metric value with high confidence.

This gives us a bound on violation probability.)

- We now want to **modify** π_{base} into a new π_{task}
 - π_{task} should maximize r_{task} while inheriting safety from π_{base}
- How to **maintain** safety guarantees?
 - Modifying sample distribution generally breaks PAC bounds...

Now suppose we want to modify our certified base policy into a new task policy which maximizes task-specific reward while inheriting safety from the base policy.

But, how can we maintain our safety guarantees?

If the policy changes, the trajectory distribution changes, which means that our PAC-based bounds would traditionally no longer apply.

However, we found a solution, under certain conditions.

- For any **modified** policy π_{task} , if the **policy ratio** is bounded:

$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha \quad \forall a \in A, s \in S$$

then the **probability of safety violation** for π_{task} is bounded:

$$P(\tau_{\pi_{task}} \not\models \varphi) \leq \epsilon_{base} \alpha^T$$

Bound on violation probability for task policy

In the paper we prove that if the policy ratio is bounded by some value alpha for all state-action pairs, then the probability of safety violation for the task policy is bounded by that of the base policy, multiplied by alpha exponentiated by the episode length.

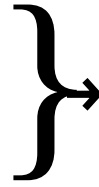
Bound on violation probability for base policy

$$P(\tau_{\pi_{base}} \not\models \varphi) \leq \epsilon_{base}$$

and

$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha \quad \forall a \in A, s \in S$$

Bound on policy ratio



Bound on violation probability for task policy

$$P(\tau_{\pi_{task}} \not\models \varphi) \leq \epsilon_{base} \alpha^T$$

No required knowledge of system dynamics, or constraints for φ !

In other words, we can obtain a prior bound on the probability of safety violation for any task policy based entirely on the base policy, with no required knowledge of the system dynamics, or the constraints of the property.

Quick Note: Extension to Robust Control



- We have thus far assumed μ and p are unchanged
- Our results can **generalize** to changes in both
 - Can be applied to **robust control settings** for perturbed systems
- See the paper for details!

Our result can also generalize to robust control, which I'm happy to talk more about in the discussion.

(We have thus far assumed that the initial state distribution and state transition distribution are unchanged

However, our results can actually generalize to changes in both.

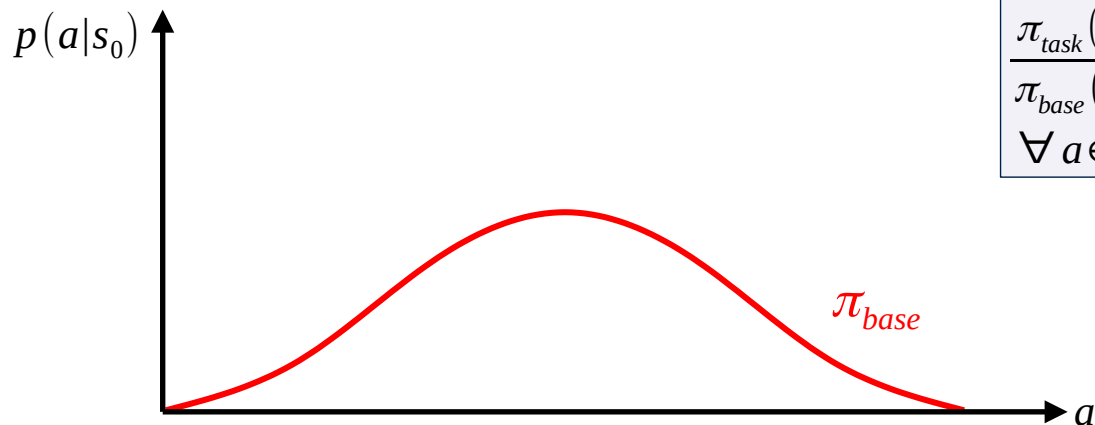
So, they can be applied to robust control settings for perturbed systems, for example.

See the paper for details.)

Property Violation under Modified Policies



$t=0$



We require:

$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

SPoRt - Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL (Cloete et. al.)

15

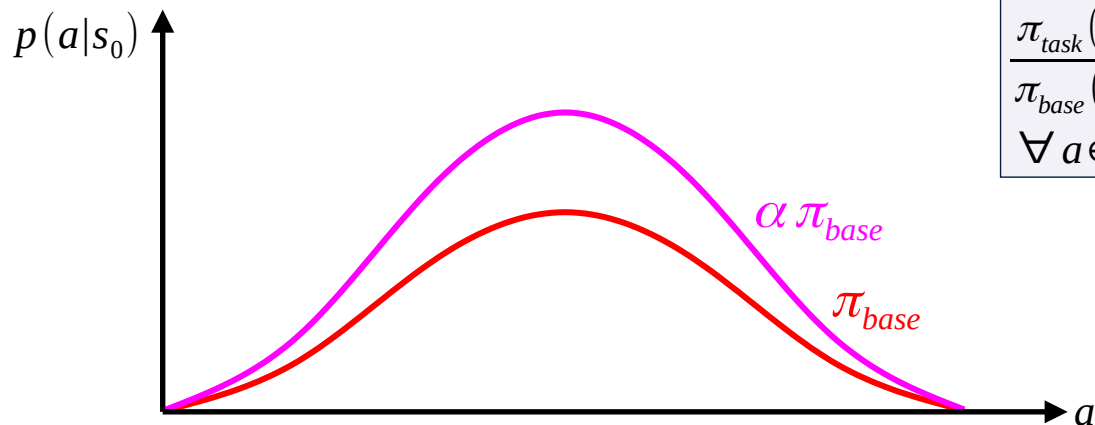
Let's visualize an episode rollout for a candidate unconstrained task policy.

Here we've got the base policy at time 0.

Property Violation under Modified Policies



$t=0$



We require:

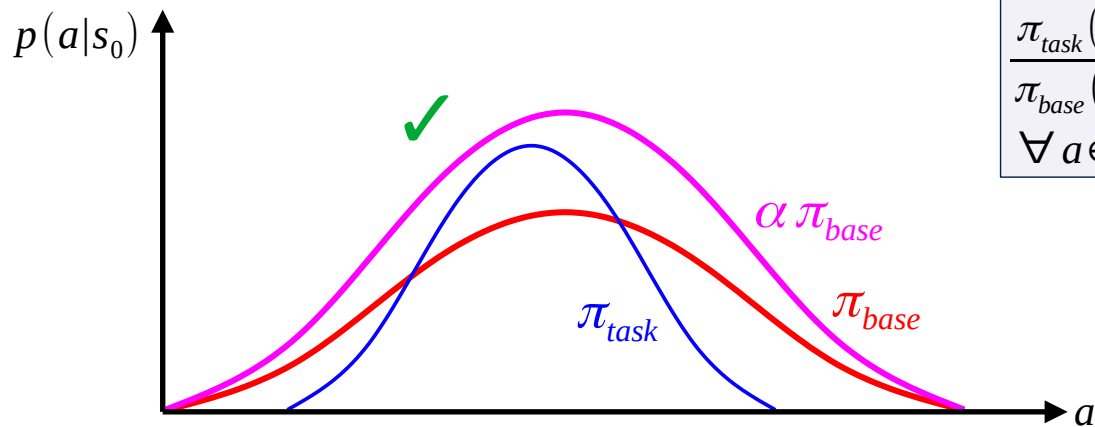
$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

Let's multiply it up by alpha. For our safety guarantees to hold, our task policy must remain below this new line.

Property Violation under Modified Policies



$t=0$



We require:

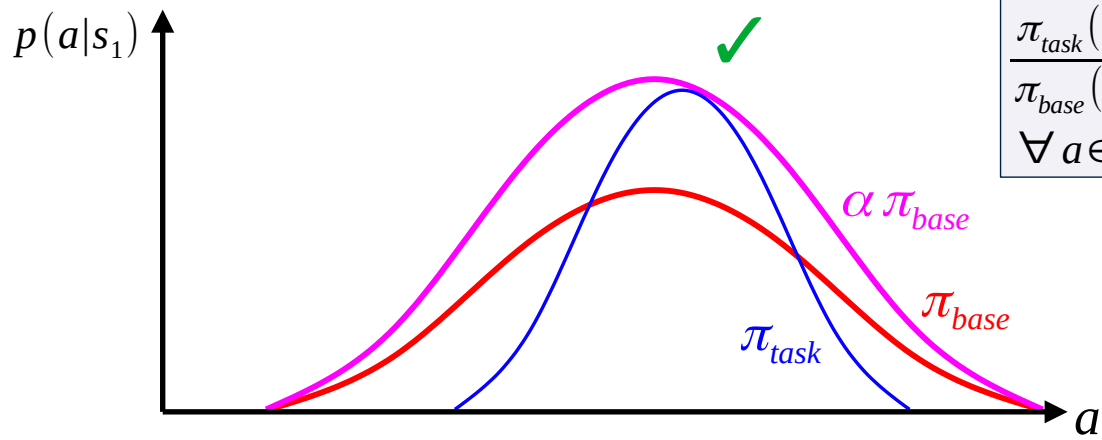
$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

And here's the unconstrained task policy. All good so far.

Property Violation under Modified Policies



$t=1$



We require:

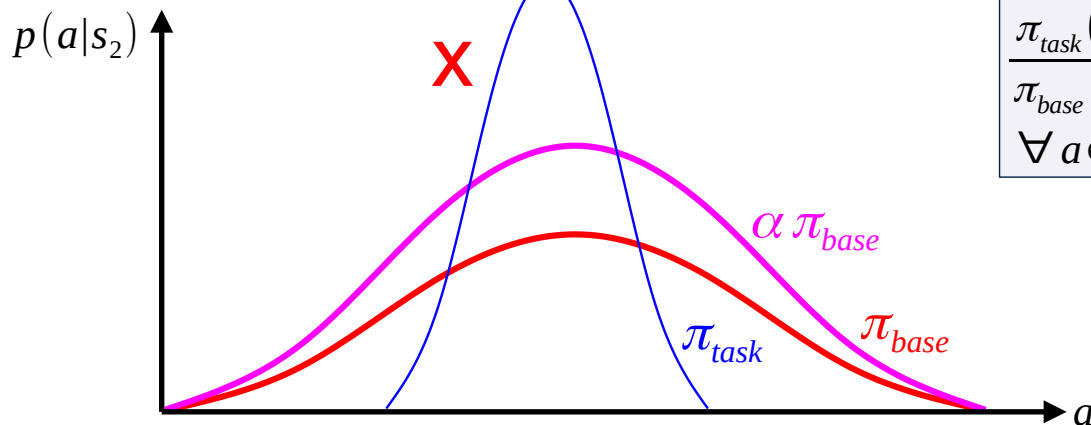
$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

Still good for time 1.

Property Violation under Modified Policies



$t=2$



We require:

$$\frac{\pi_{\text{task}}(a|s)}{\pi_{\text{base}}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

SPoRt - Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL (Cloete et. al.)

19

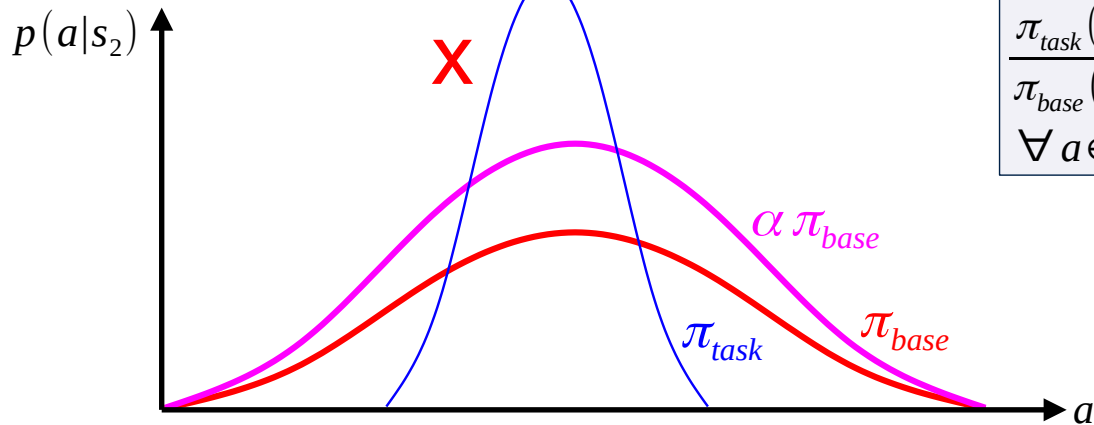
Ahh, at time 2 we see that the unconstrained task policy violates the bound on the policy ratio.

This is a good time to point out that the safety of our task policy relies on it being in the support of the base policy;

if we try to deviate too much, our bound becomes huge, since we're going way out of distribution.

Property Violation under Modified Policies

$t=2$



We require:

$$\frac{\pi_{\text{task}}(a|s)}{\pi_{\text{base}}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

How can we ensure the policy ratio is bounded for all state-action pairs?

Now, in order for our safety guarantees to hold, we need to ensure that the policy ratio is always bounded by alpha, regardless of what state the agent is in.

But how can we do this?

At each time step...

We require:

$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

π_{base}
★
($\alpha=1$)

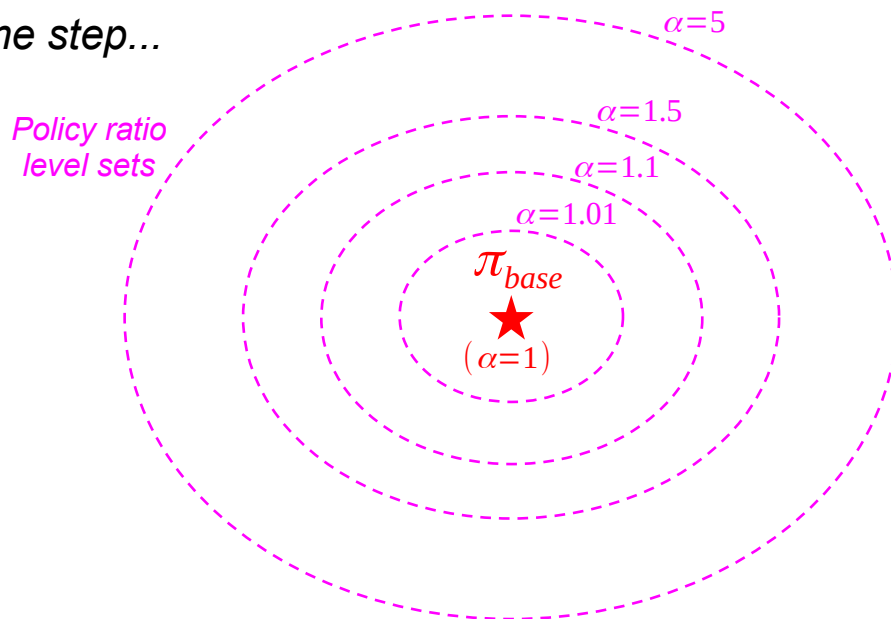
(Policy space Π)

Let's visualize our base policy in “policy space” at a given time step.

For a diagonal Gaussian policy, the dimensions of this space would be the means and variances.

Policy Projection

At each time step...



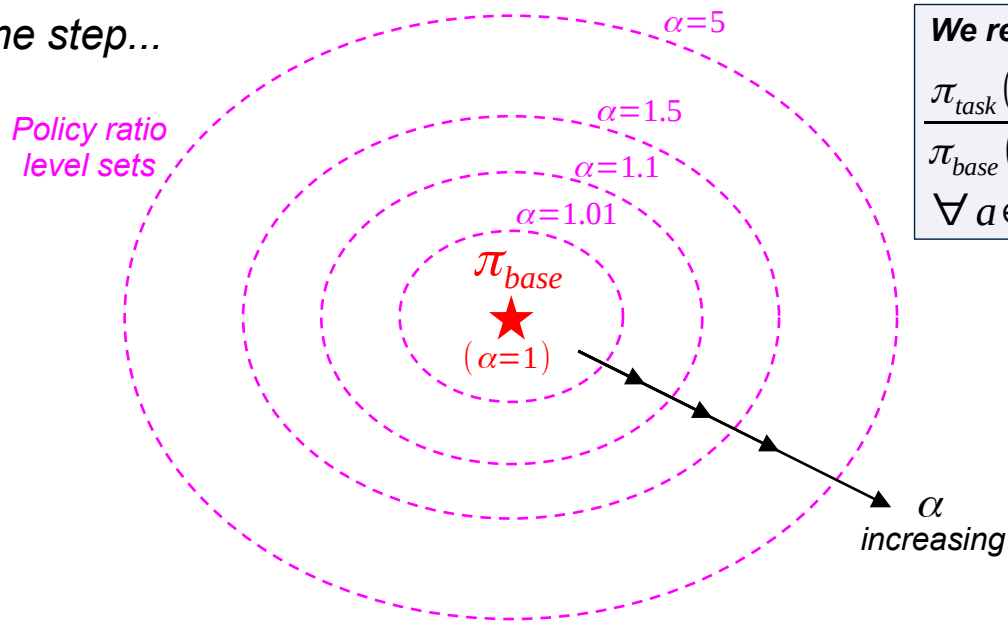
We require:

$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

For different values of alpha we can draw level sets of policies where the policy ratio is less than or equal to alpha for all actions.

Policy Projection

At each time step...



We require:

$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

SPoRt - Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL (Cloete et. al.)

23

And notice that as alpha increases, the level sets become larger.

This makes sense; we're allowing our bound on safety violation to go up, but we gain more flexibility for what our task policy can be.

$$P(\tau_{\pi_{\text{task}}} \neq \varphi) \leq \epsilon_{\text{base}} \alpha^T \leq \epsilon_{\text{max}}$$

Maximum acceptable violation probability

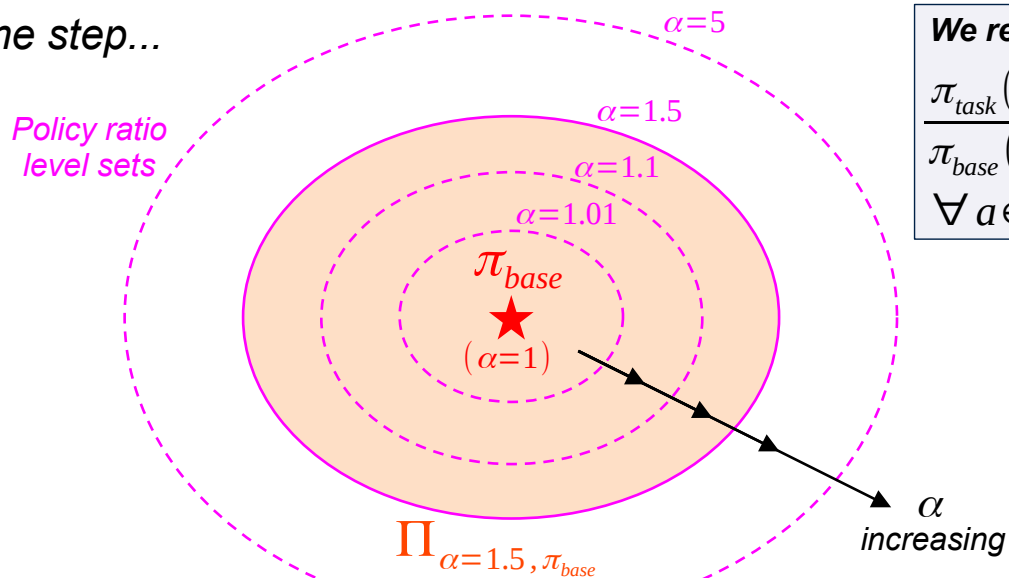
$$\Rightarrow \alpha \leq \sqrt[T]{\frac{\epsilon_{\text{max}}}{\epsilon_{\text{base}}}}$$

Suppose $\epsilon_{\text{base}} = 0.00017$, $\epsilon_{\text{max}} = 0.01$, $T = 10 \Rightarrow \alpha = 1.5$

A good way to choose alpha is to first decide on a maximum acceptable violation probability and then rearrange our bound to recover maximum allowed alpha.

Policy Projection

At each time step...



We require:

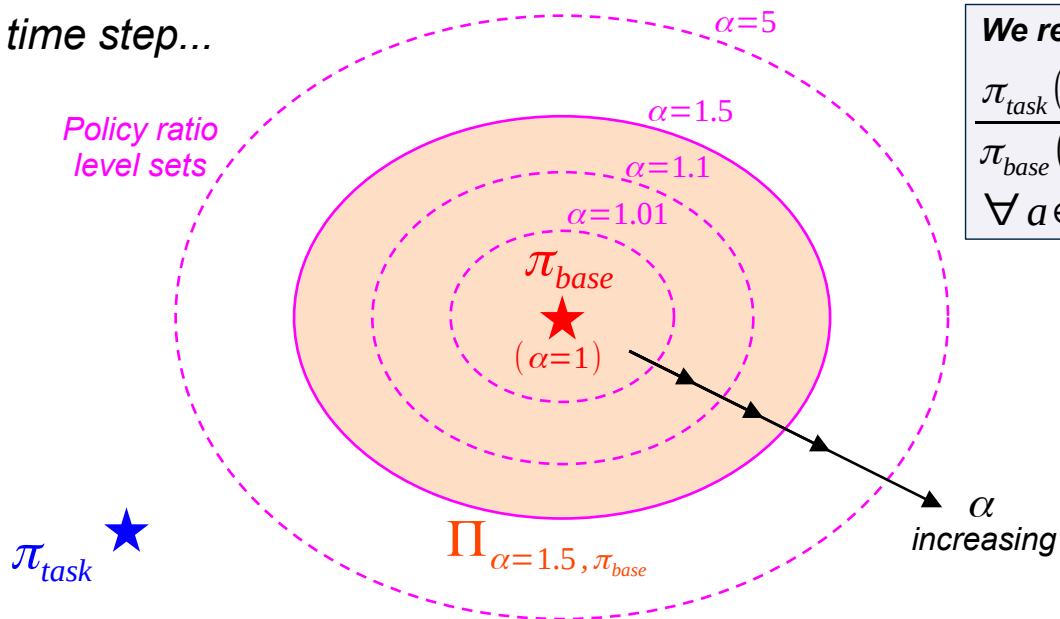
$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$

$$\forall a \in A, s \in S$$

With our chosen alpha, we now have a feasible set of allowed task policies.

Policy Projection

At each time step...



We require:

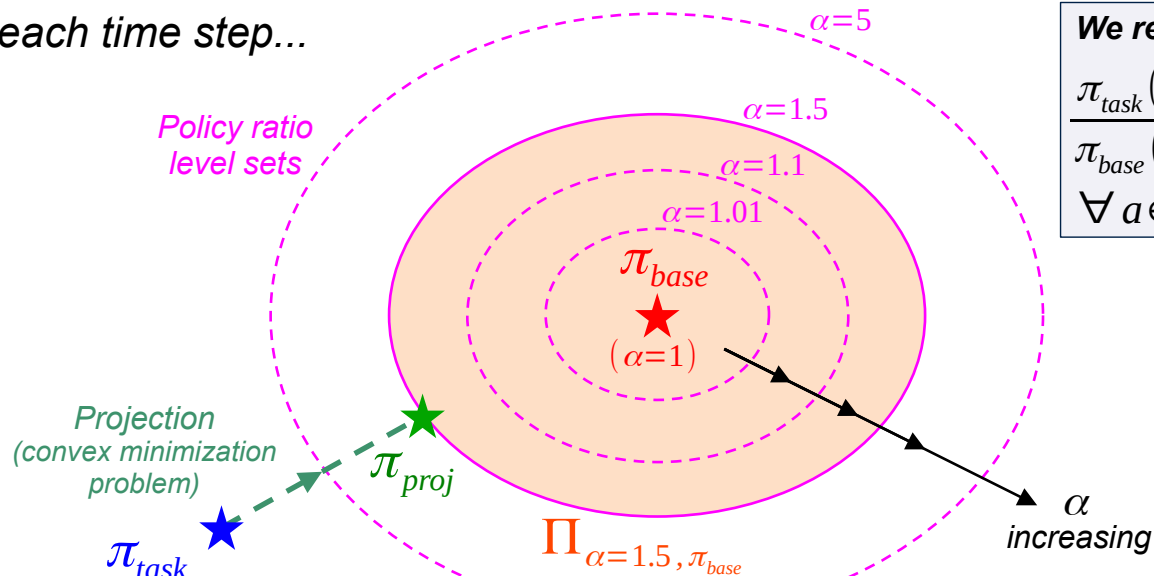
$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$

$$\forall a \in A, s \in S$$

Suppose our unconstrained task policy lies outside this feasible set at a given time step.

Policy Projection

At each time step...



We require:

$$\frac{\pi_{task}(a|s)}{\pi_{base}(a|s)} \leq \alpha$$
$$\forall a \in A, s \in S$$

SPoRt - Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL (Cloete et. al.)

27

The trick is to use projection.

For diagonal Gaussian policies, we can project an unconstrained task policy onto this feasible set as a convex minimization problem.

The projection minimizes KL divergence between the task policy and our new projected policy.

We then sample our action from the projected policy.

In this way, the bound on the policy ratio will always hold.

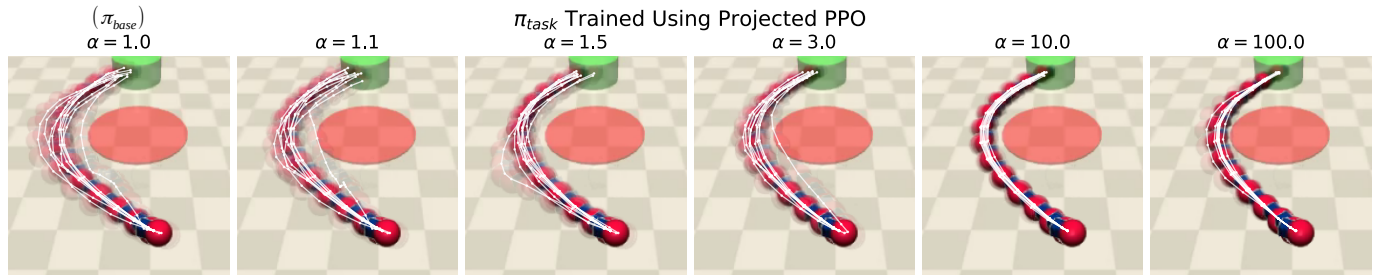
- **Clipped PPO**, but with policy projection at every time step
 - Requires π_{base} , ϵ_{base} , α and a new **task-specific reward function**
 - Train policy network for **unconstrained** π_{task} (initialized as π_{base})
 - Always sample from π_{proj} during **both training and inference**

So, how should we train the task policy, given a new task-specific reward function?

The answer is to use Projected PPO, which is clipped PPO but with policy projection at every time step.

We train the policy network for the unconstrained task policy (initialized as the base policy) to maximize the task-specific reward, but always sample from the projected policy, during both training and inference.

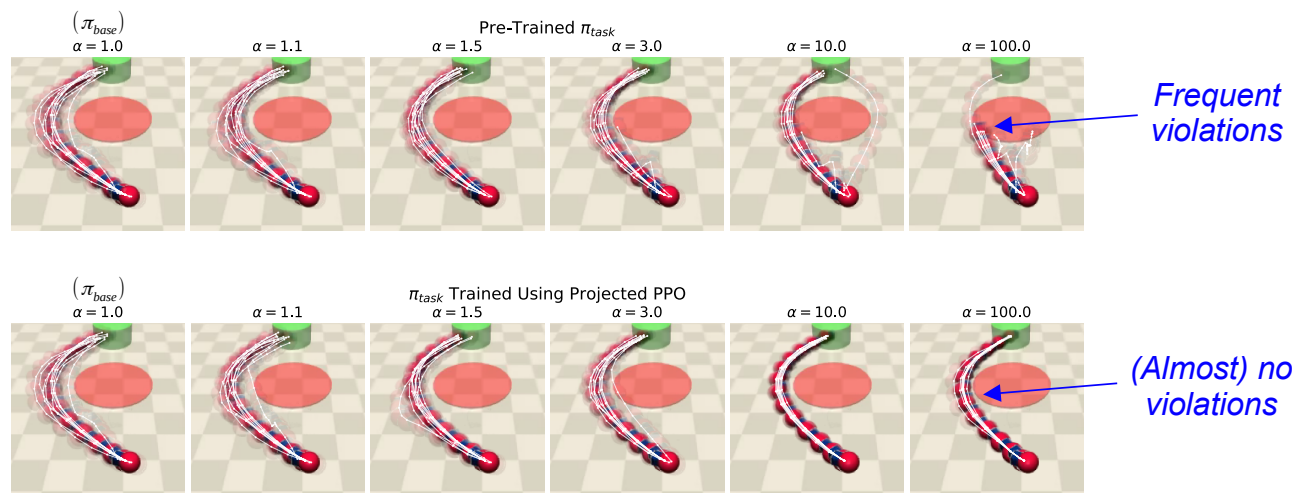
- Applied to a **reach-avoid** safety property
 - Task-specific reward function to **minimize time to reach goal**
(No consideration of hazard in task-specific reward)



We applied SPoRt to a reach-avoid safety property for a point agent, with a task-specific reward function to minimize time to reach the goal.

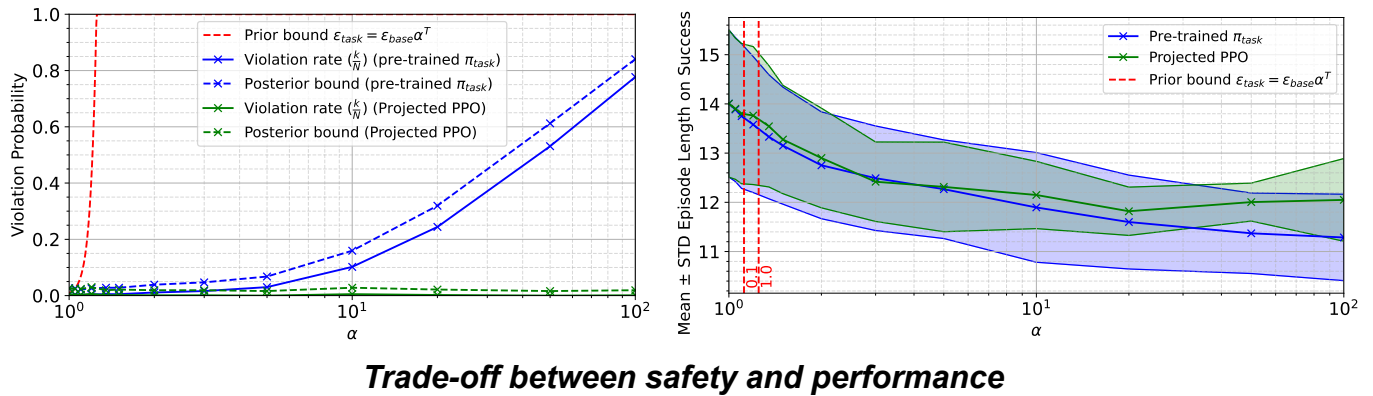
As you can see, as alpha increases, the agent turns more tightly around the hazard, reaching the goal faster but getting closer to safety violation.

Results



Projected PPO also outperforms naïvely projecting an unsafe pre-trained task policy at inference time to enforce safety.

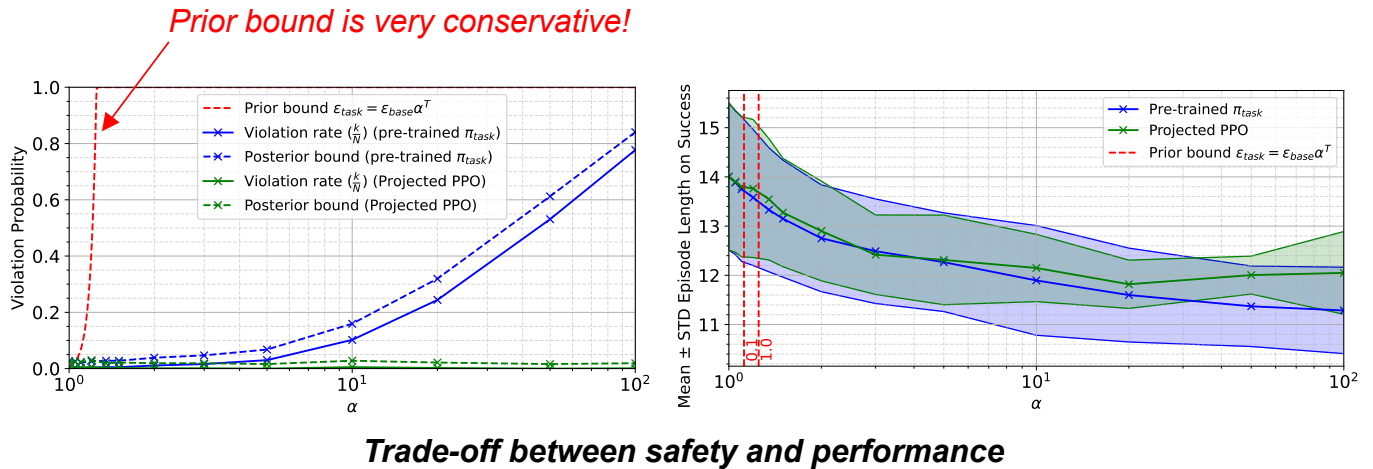
Results



As alpha increases, time taken on success decreases while violation risk increases.

Thus, we see a trade-off between safety and performance.

Results



SPoRt - Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL (Cloete et. al.)

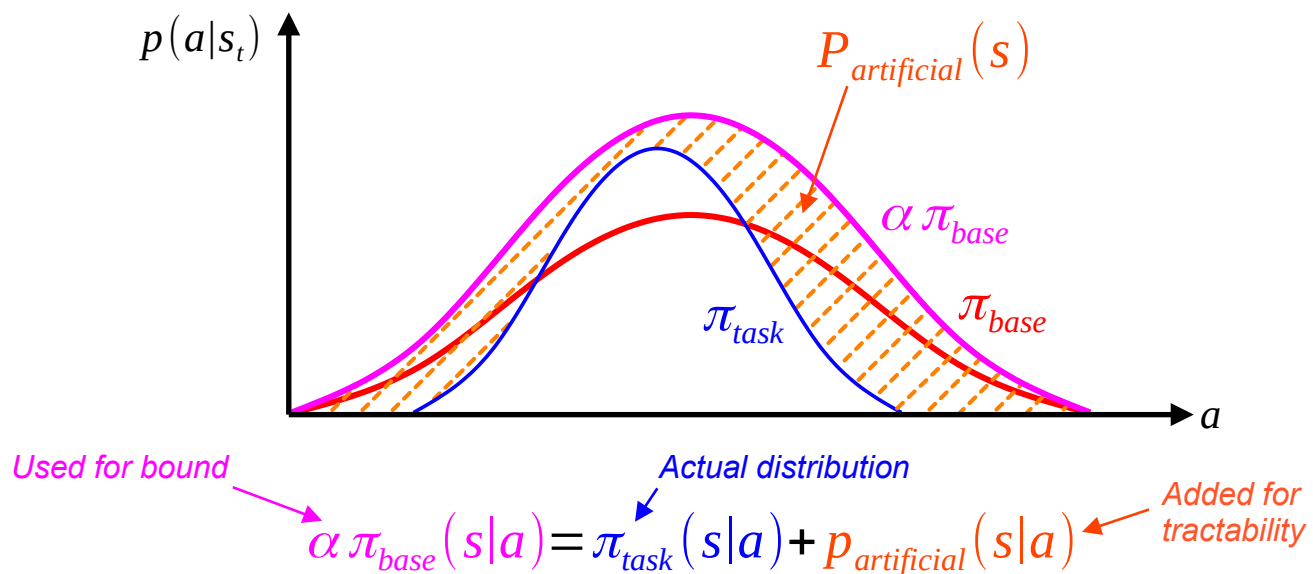
32

However, SPoRt is not without limitations, the biggest being that the prior bound can be very conservative for large α and episode length.

Future work ought to improve upon this conservativeness to make it more practically useful.

In the meantime, α could be treated more like a hyperparameter for risk-aversion.

Conservativeness of the Prior Bound



SPoRt - Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL (Cloete et. al.)

33

I'm happy to talk more about the nature of the conservativeness in the discussion.

(The biggest contributor to the conservativeness is the “artificial probability mass” that we add to the bound to make it tractable.

The base policy distribution scaled by alpha, which we use to construct the bound, is the sum of the task policy distribution and the artificial probability distribution.

The artificial probability mass makes the bound “think” we’re more likely to take each action than we actually are, including actions that generate unsafe trajectories, hence raising the bound on safety violation.)

Thank You for Listening!



Full paper here:



arxiv.org/abs/2504.06386

Contact:



 jacques@robots.ox.ac.uk

 [jacquescloete.github.io](https://github.com/jacquescloete)

 [jacques-cloete](https://www.linkedin.com/in/jacques-cloete)

 [JacquesCloete](https://github.com/JacquesCloete)

This work was supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems [EP/S024050/1]



Engineering and
Physical Sciences
Research Council



SPoRt - Safe Policy Ratio: Certified Training and Deployment of Task Policies in Model-Free RL (Cloete et. al.)

34

That's all, thank you for listening!