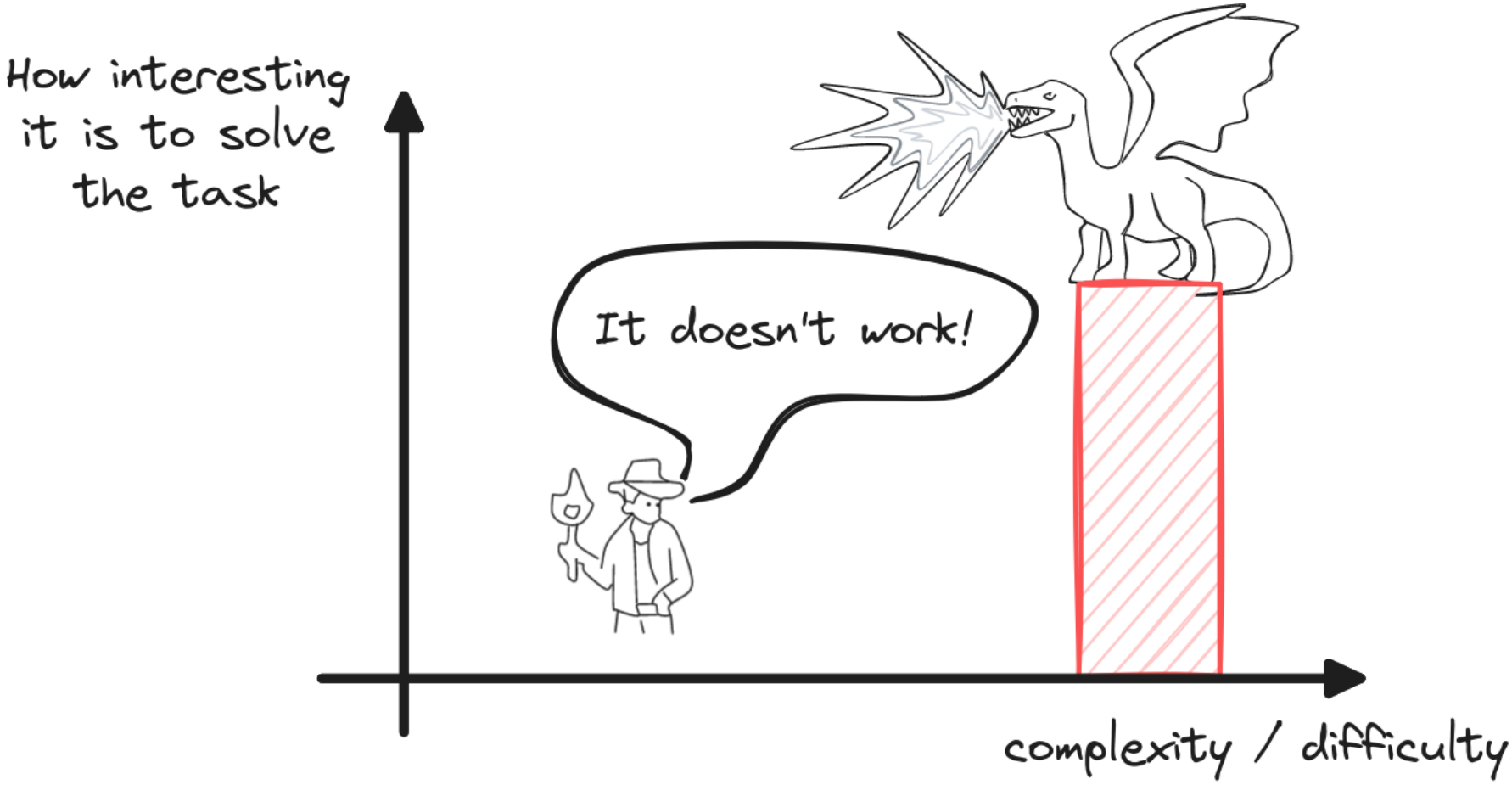
DESIGNING AND RUNNING REAL-WORLD REINFORCEMENT LEARNING EXPERIMENTS

Antonin RAFFIN (@araffin2)
German Aerospace Center (DLR)
https://araffin.github.io/

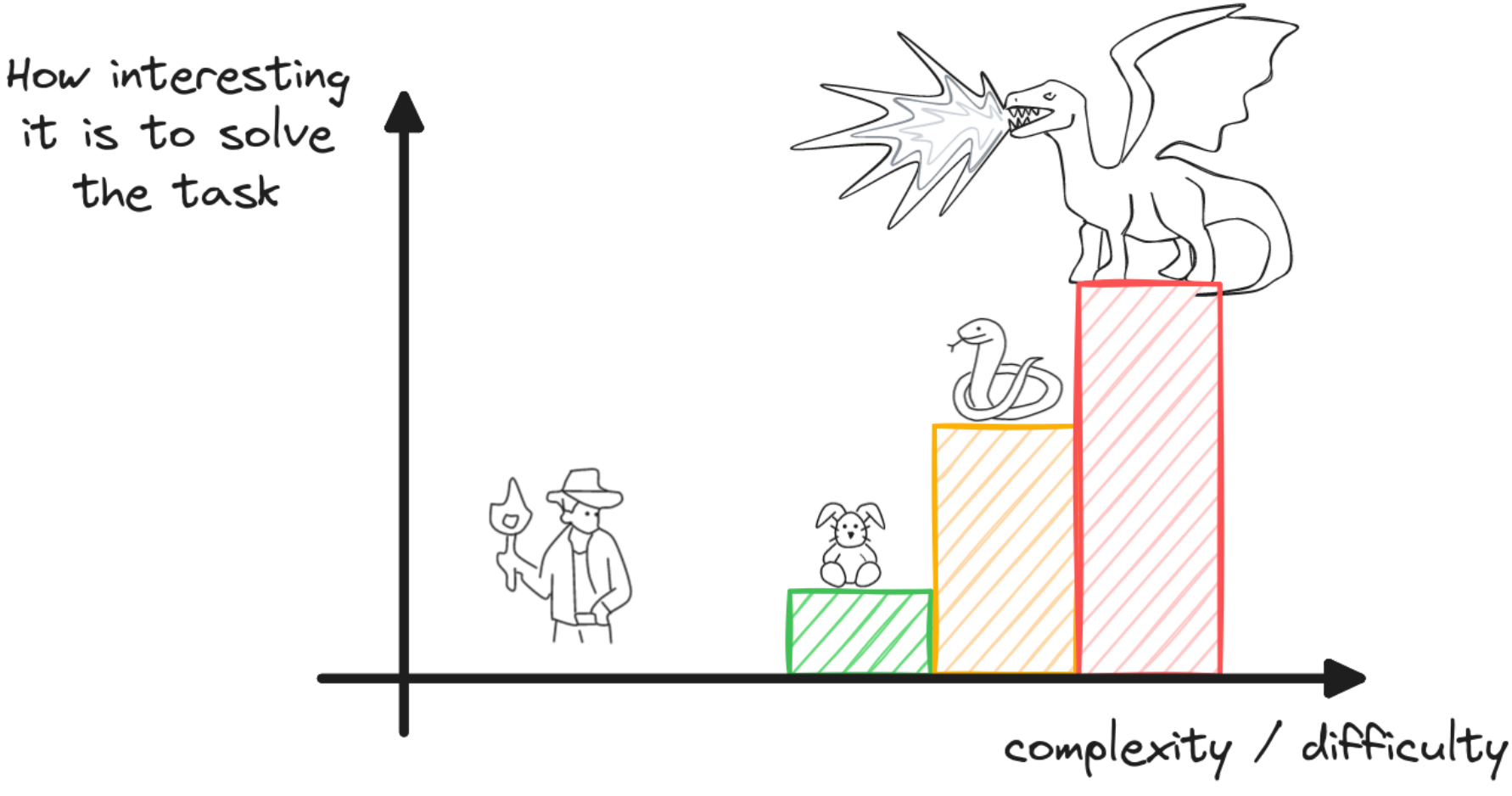
BORING PROBLEMS ARE IMPORTANT

BORING PROBLEMS ARE IMPORTANT



START SIMPLE!

START SIMPLE!

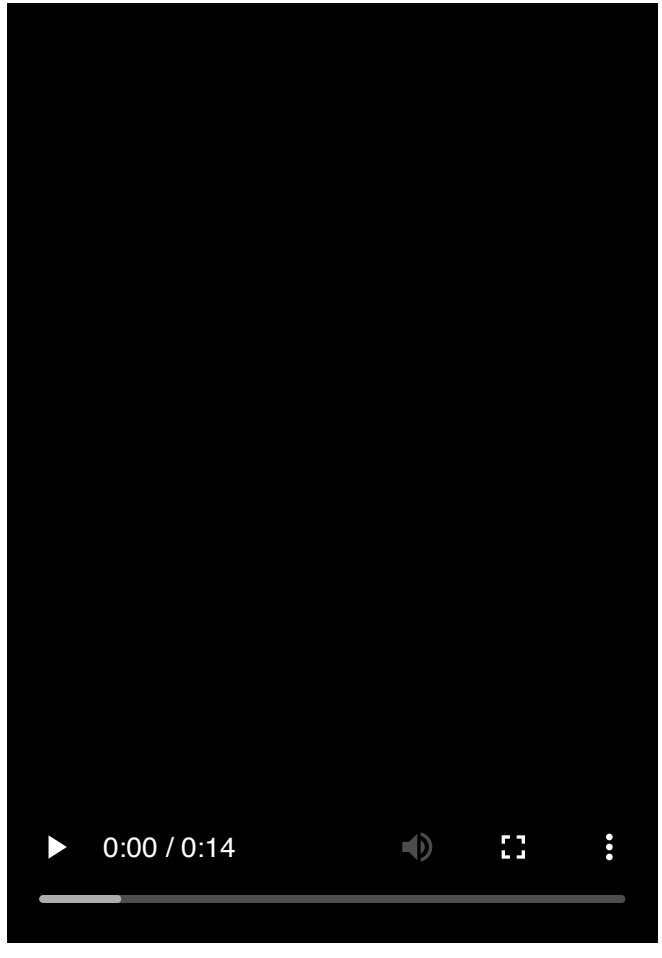


MOTIVATION

Learning directly on real robots

SIMULATION IS ALL YOU NEED?

SIMULATION IS ALL YOU NEED?



ISS EXPERIMENT (1)

Credit: ESA/NASA

ISS EXPERIMENT (2)

ISS EXPERIMENT (2)

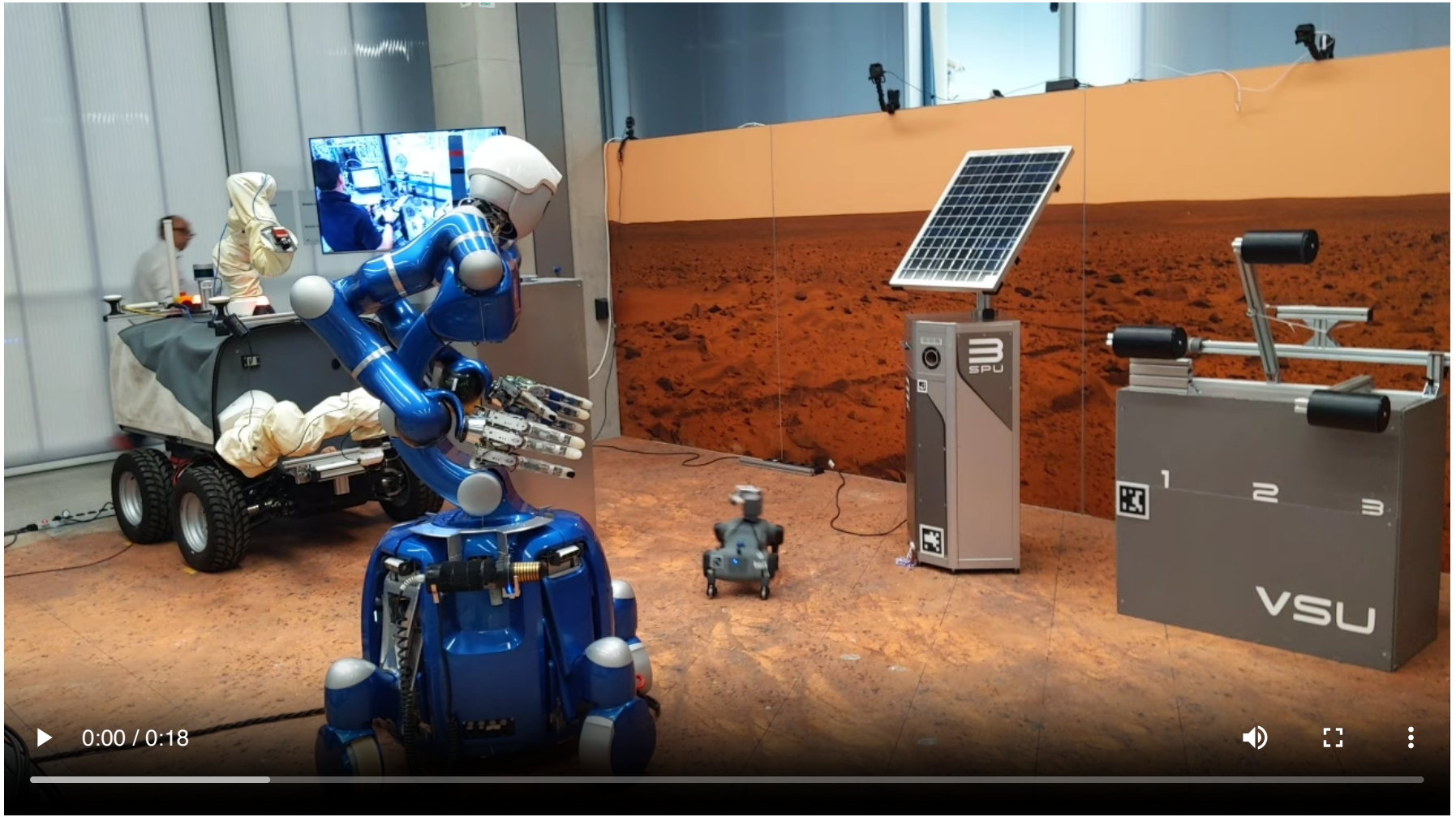
Before

ISS EXPERIMENT (2)

Before

After, with the 1kg arm

CAN IT TURN?

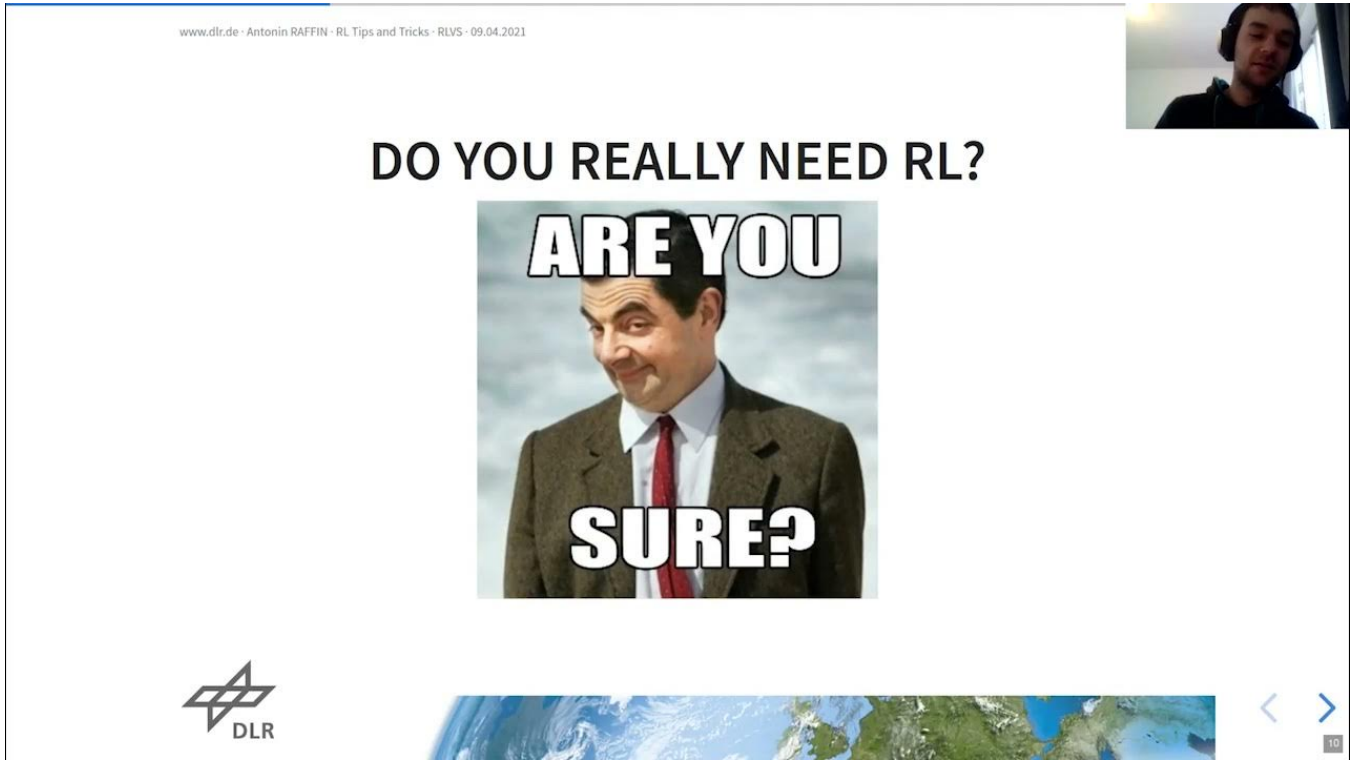


ADDITIONAL VIDEO

OUTLINE

1. **Task design**
2. Choosing an algorithm
3. Safety layers
4. Running the experiments
5. Troubleshooting

RL IN PRACTICE: TIPS AND TRICKS - VIDEO



TASK DESIGN

TASK DESIGN

- Observation space

TASK DESIGN

- Observation space
- Action space

TASK DESIGN

- Observation space
- Action space
- Reward function

TASK DESIGN

- Observation space
- Action space
- Reward function
- Termination conditions

TRUNCATIONS FOR INFINITE HORIZON TASKS

truncation vs termination

TRUNCATIONS FOR INFINITE HORIZON TASKS

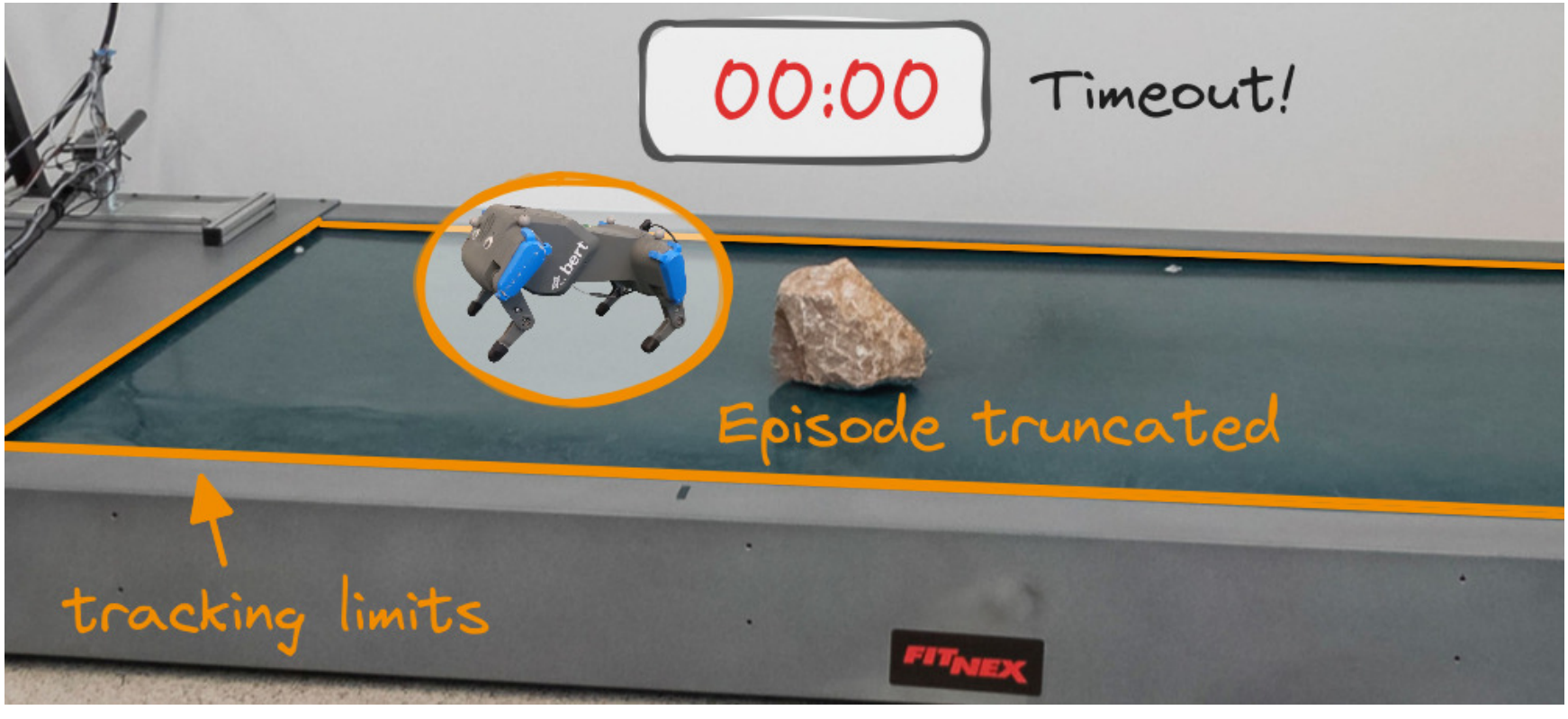
truncation vs termination

TRUNCATIONS FOR INFINITE HORIZON TASKS

truncation vs termination

TRUNCATIONS FOR INFINITE HORIZON TASKS

truncation vs termination



TRUNCATIONS FOR INFINITE HORIZON TASKS

truncation vs termination

EXAMPLE

EXAMPLE

$$\forall t, \quad r_t = 1, \quad \gamma = 0.98$$

EXAMPLE

$$\forall t, \quad r_t = 1, \quad \gamma = 0.98$$

Timeout: `max_episode_steps=4`

EXAMPLE

$$\forall t, \quad r_t = 1, \quad \gamma = 0.98$$

Timeout: `max_episode_steps=4`

- Without truncation handling:
  $V_\pi(s_0) = \sum_{t=0}^{3}[\gamma^t r_t] = 1 + 1 \cdot 0.98 + 0.98^2 + 0.98^3 \approx 3.9$

EXAMPLE

$$\forall t, \quad r_t = 1, \quad \gamma = 0.98$$

Timeout: `max_episode_steps=4`

- Without truncation handling:
  $V_\pi(s_0) = \sum_{t=0}^{3}[\gamma^t r_t] = 1 + 1 \cdot 0.98 + 0.98^2 + 0.98^3 \approx 3.9$

- With truncation handling:
  $V_\pi(s_0) = \sum_{t=0}^{\infty}[\gamma^t r_t] = \sum_{t=0}^{\infty}[\gamma^t] = \frac{1}{1-\gamma} \approx 50$

RECALL: DQN UPDATE

RECALL: DQN UPDATE

1. DQN loss:

$$\mathcal{L} = \mathbb{E}[(y_t - Q_\theta(s_t, a_t))^2]$$

RECALL: DQN UPDATE

1. DQN loss:

$$\mathcal{L} = \mathbb{E}[(y_t - Q_\theta(s_t, a_t))^2]$$

2. Regression $f_\theta(x) = y$ with input x and target y:

RECALL: DQN UPDATE

1. DQN loss:

$$\mathcal{L} = \mathbb{E}[(y_t - Q_\theta(s_t, a_t))^2]$$

2. Regression $f_\theta(x) = y$ with input x and target y:

- input: $x = (s_t, a_t)$

RECALL: DQN UPDATE

1. DQN loss:

$$\mathcal{L} = \mathbb{E}[(y_t - Q_\theta(s_t, a_t))^2]$$

2. Regression $f_\theta(x) = y$ with input x and target y:

- input: $x = (s_t, a_t)$
- if s_{t+1} is **non** terminal: $y = r_t + \gamma \cdot \max_{a' \in A}(Q_\theta(s_{t+1}, a'))$

RECALL: DQN UPDATE

1. DQN loss:

$$\mathcal{L} = \mathbb{E}[(y_t - Q_\theta(s_t, a_t))^2]$$

2. Regression $f_\theta(x) = y$ with input x and target y:

- input: $x = (s_t, a_t)$
- if s_{t+1} is **non** terminal: $y = r_t + \gamma \cdot \max_{a' \in A}(Q_\theta(s_{t+1}, a'))$
- if s_{t+1} is terminal: $y = r_t$

RECALL: DQN UPDATE

1. DQN loss:

$$\mathcal{L} = \mathbb{E}[(y_t - Q_\theta(s_t, a_t))^2]$$

2. Regression $f_\theta(x) = y$ with input x and target y:

- input: $x = (s_t, a_t)$
- if s_{t+1} is **non** terminal: $y = r_t + \gamma \cdot \max_{a' \in A}(Q_\theta(s_{t+1}, a'))$
- if s_{t+1} is terminal: $y = r_t$
- **if s_{t+1} is truncation**: $y = r_t + \gamma \cdot \max_{a' \in A}(Q_\theta(s_{t+1}, a'))$

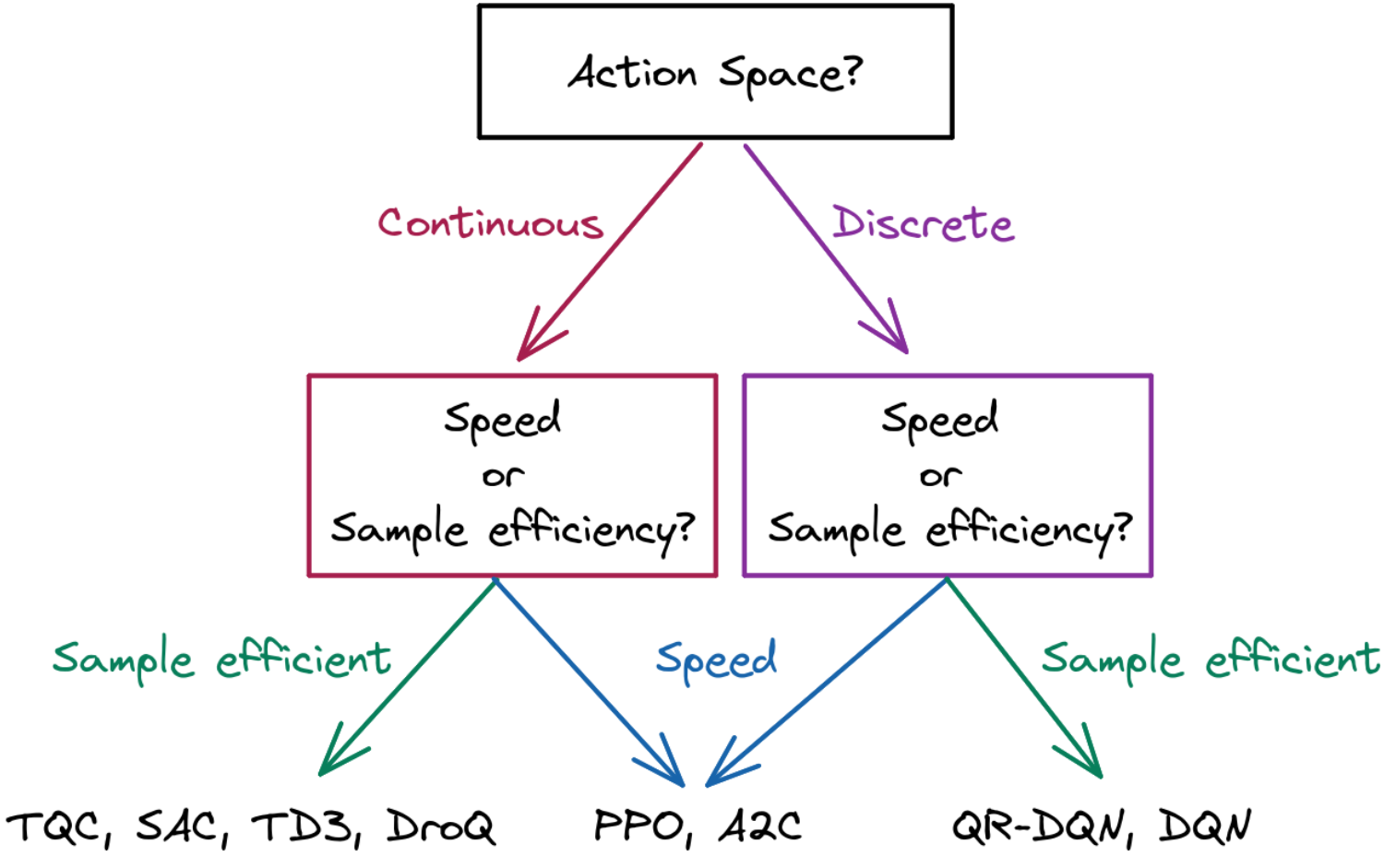
IN PRACTICE

```python
# Note: done = terminated or truncated
obs, reward, terminated, truncated, info = env.step(action)

# Offpolicy algorithms
# If the episode is terminated, set the target to be the terminal reward
should_bootstrap = np.logical_not(replay_data.terminateds)
# 1-step TD target
td_target = replay_data.rewards + should_bootstrap * (gamma * next_q_values)

# On-policy algorithms
if truncated:
    terminal_reward += gamma * next_value
```

QUESTIONS?

1. Task design
2. **Choosing an algorithm**
3. Safety layers
4. Running the experiments
5. Troubleshooting

WHICH ALGORITHM TO CHOOSE?



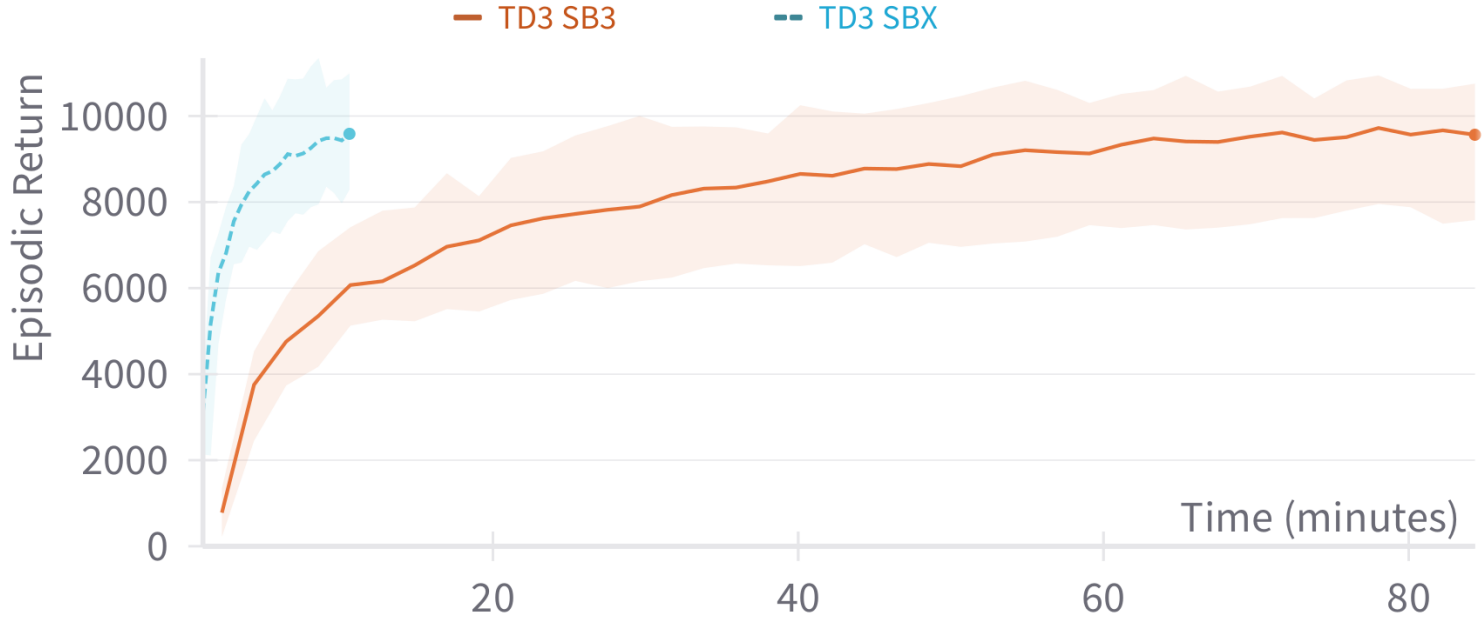
RECENT ADVANCES: JAX AND JIT

Up to 20x faster!

Stable-Baselines3 (PyTorch) vs SBX (Jax)

RECENT ADVANCES: JAX AND JIT

Up to 20x faster!



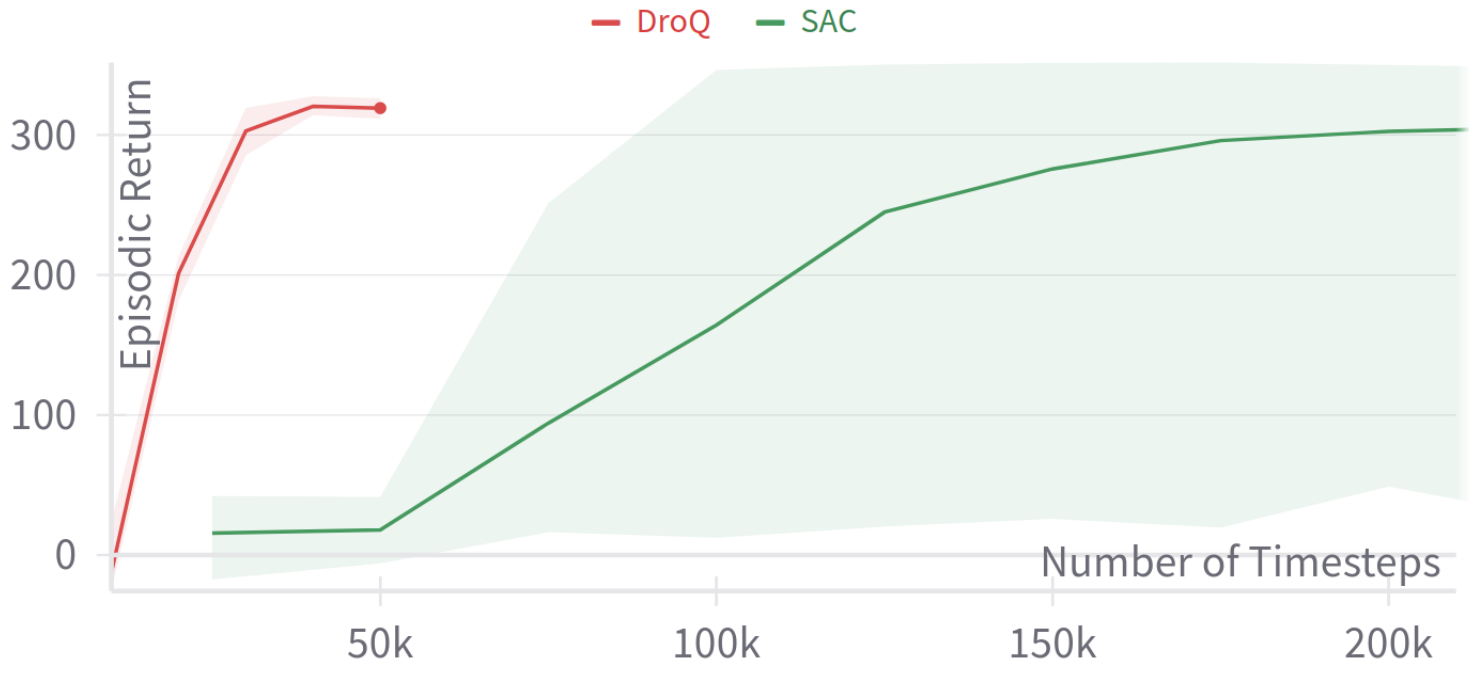
Stable-Baselines3 (PyTorch) vs SBX (Jax)

RECENT ADVANCES: DroQ

More gradient steps: 4x more sample efficient!

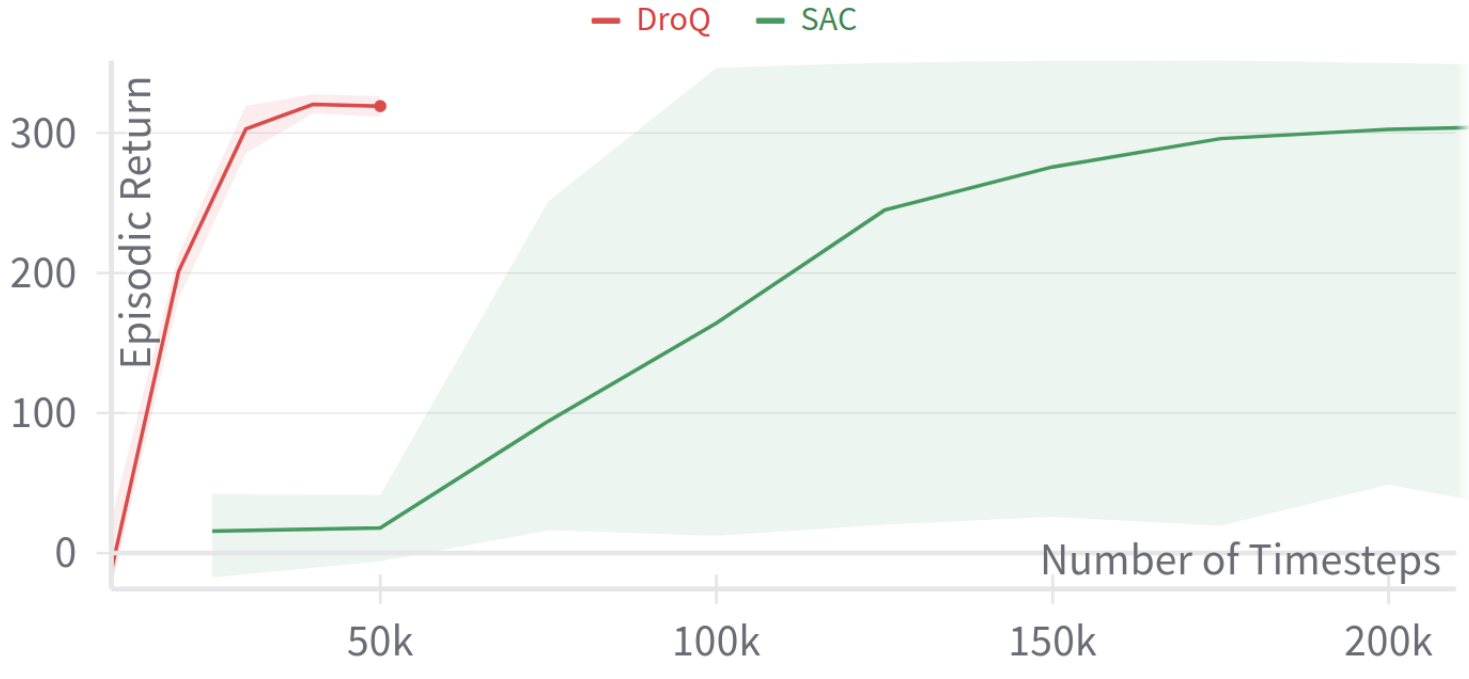
RECENT ADVANCES: DroQ

More gradient steps: 4x more sample efficient!



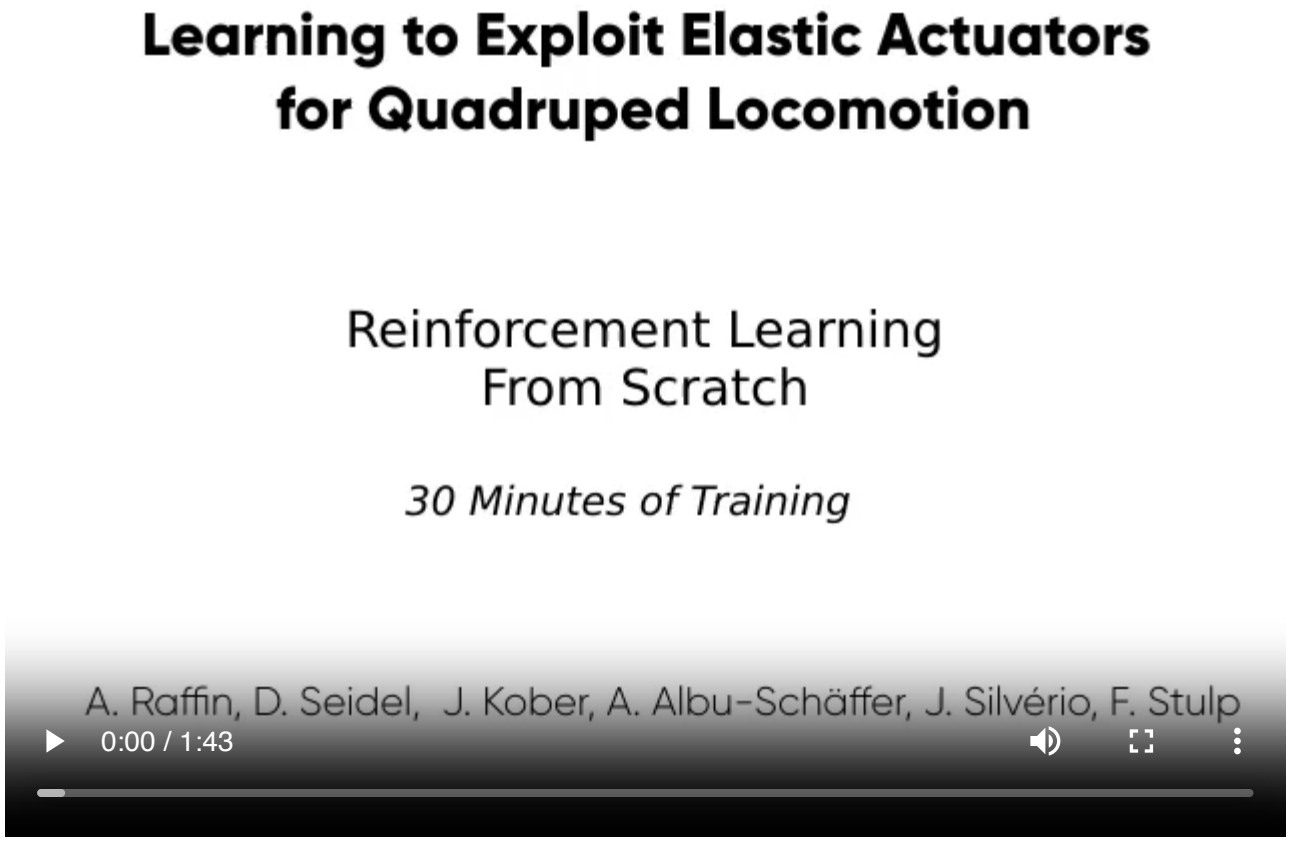
RECENT ADVANCES: DroQ

More gradient steps: 4x more sample efficient!



Also have a look at TQC, TD7 and CrossQ.

RL FROM SCRATCH IN 10 MINUTES



Using SB3 + Jax = SBX: https://github.com/araffin/sbx

QUESTIONS?

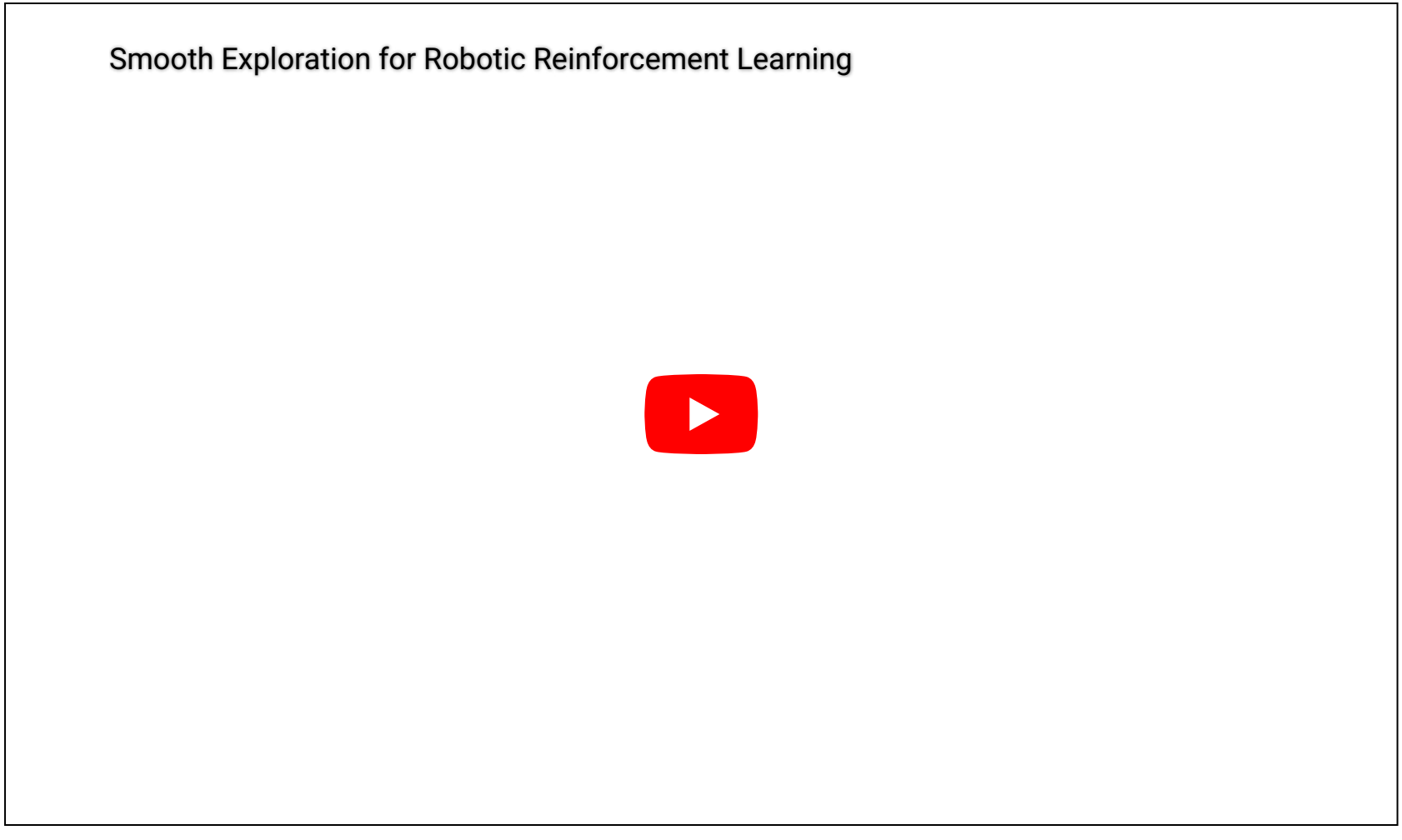
1. Task design
2. Choosing an algorithm
3. **Safety layers**
4. Running the experiments
5. Troubleshooting

HOW NOT TO A BREAK A ROBOT?

1. TASK DESIGN (ACTION SPACE)

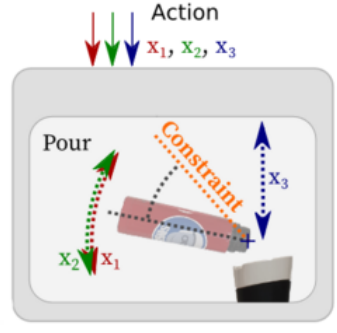
Ex: Controlling tendons forces instead of motor positions

ELASTIC NECK



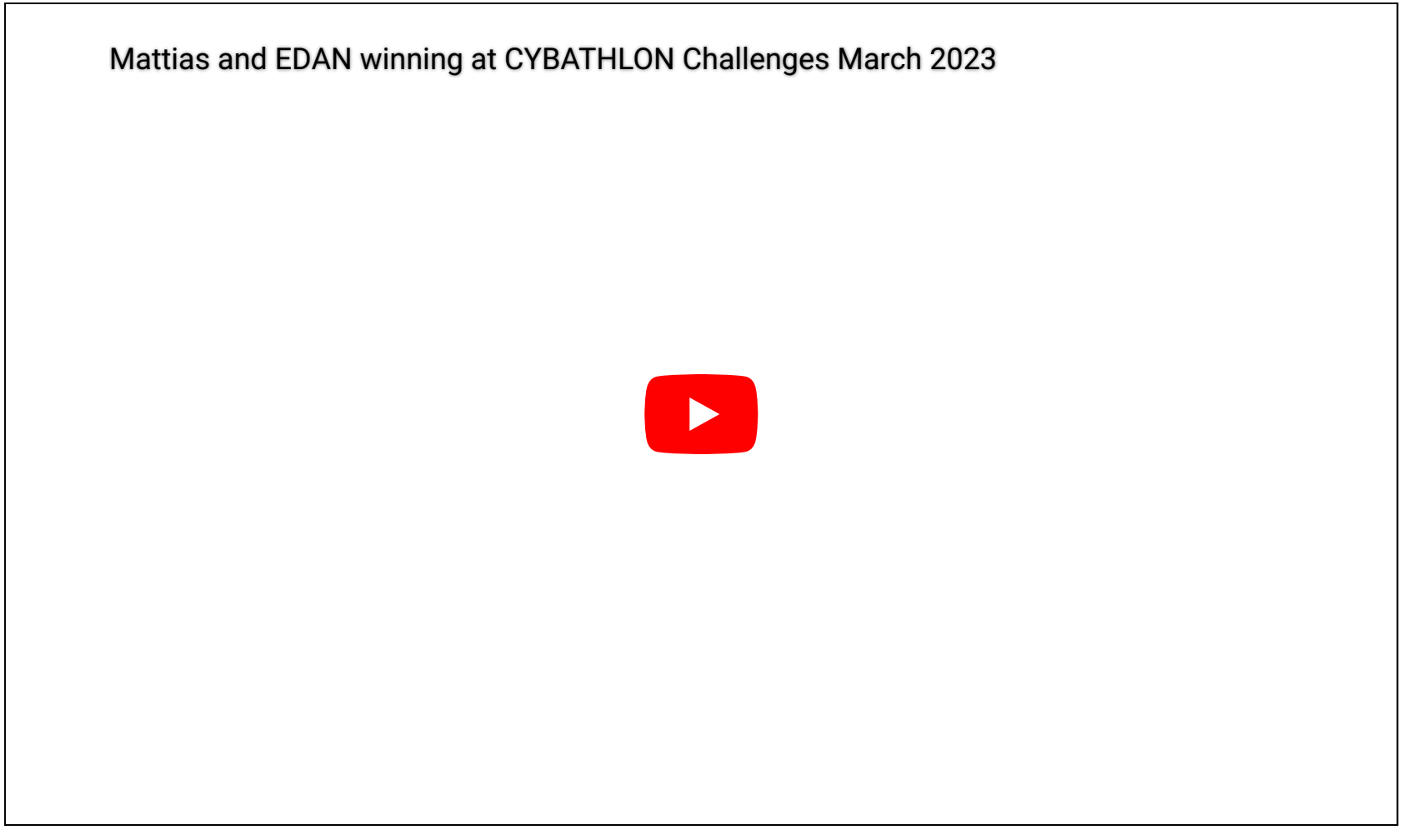
Raffin, Antonin, Jens Kober, and Freek Stulp. "Smooth exploration for robotic reinforcement learning." CoRL. PMLR, 2022.

2. HARD CONSTRAINTS, SAFETY LAYERS



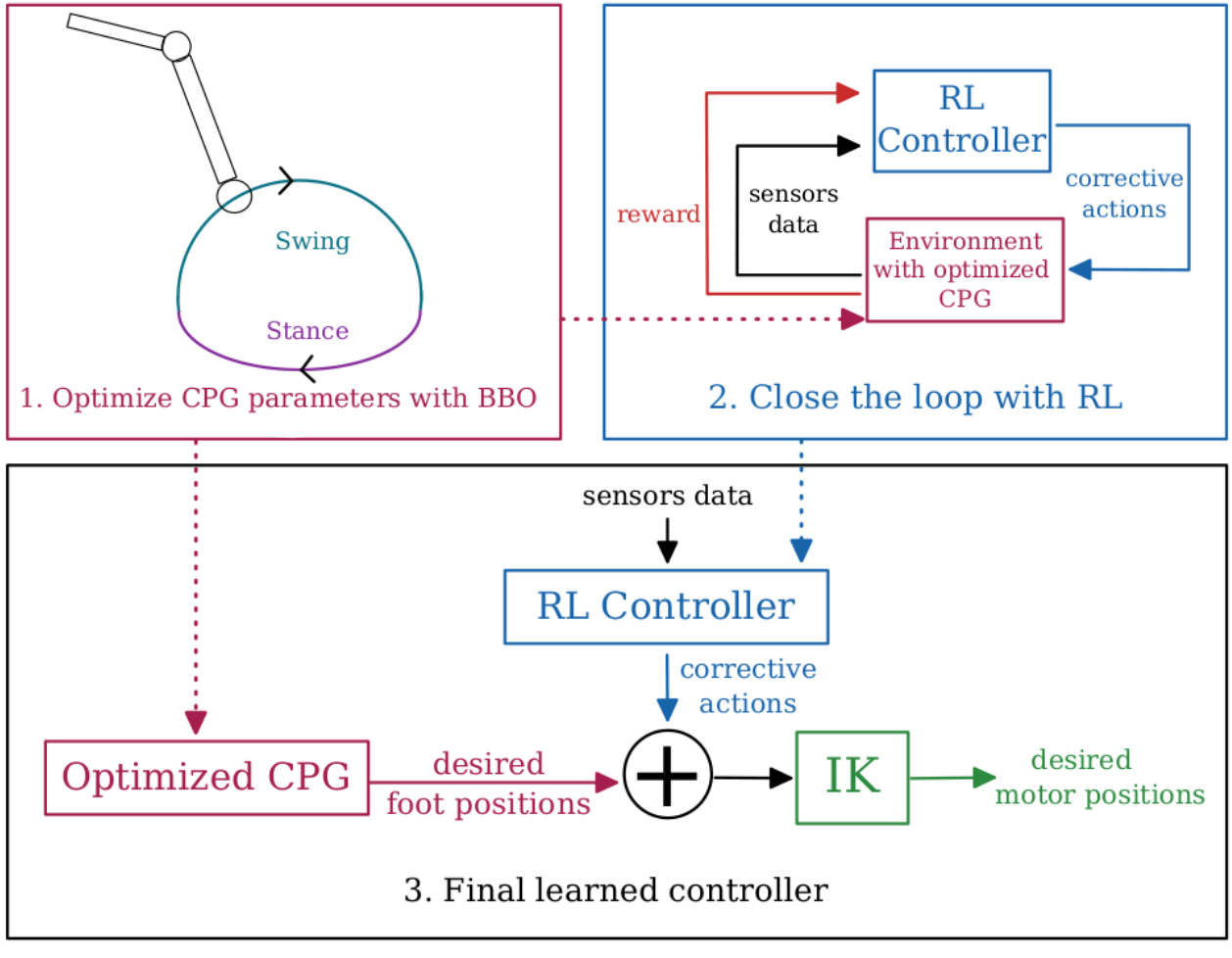
Padalkar, Abhishek, et al. "Guiding Reinforcement Learning with Shared Control Templates." ICRA 2023.

CYBATHLON CHALLENGE

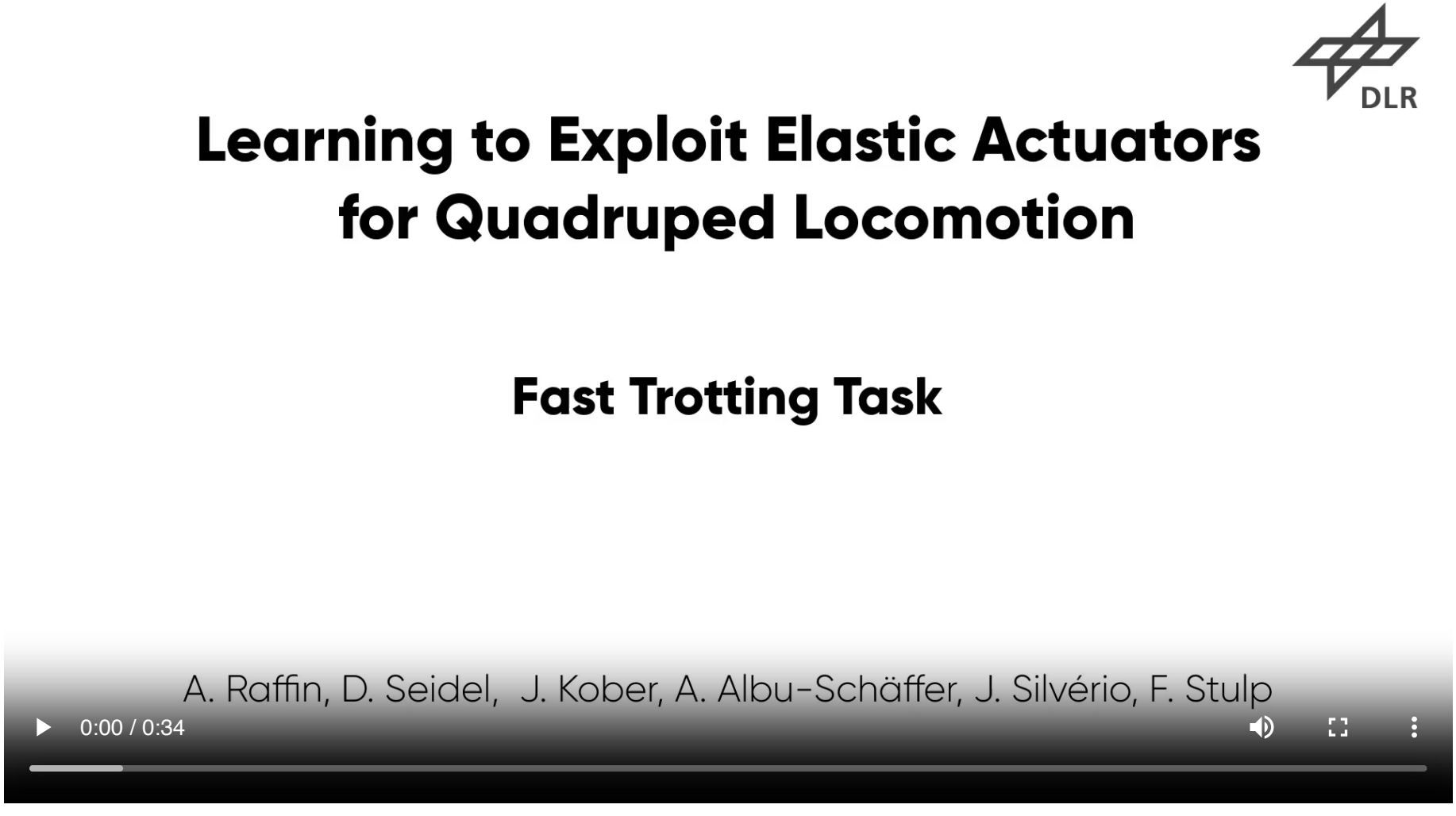


Quere, Gabriel, et al. "Shared control templates for assistive robotics." ICRA, 2020.

3. LEVERAGE PRIOR KNOWLEDGE

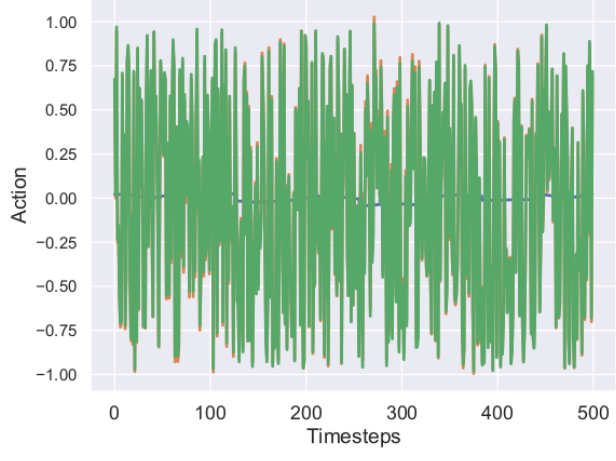


LEARNING TO EXPLOIT ELASTIC ACTUATORS

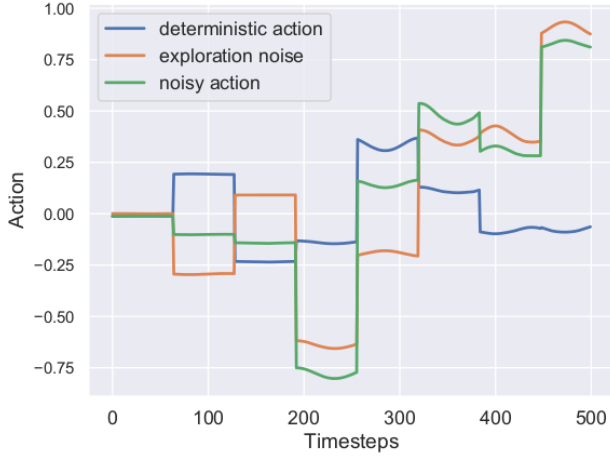


Raffin et al. "Learning to Exploit Elastic Actuators for Quadruped Locomotion" In preparation, 2023.

4. SAFER EXPLORATION

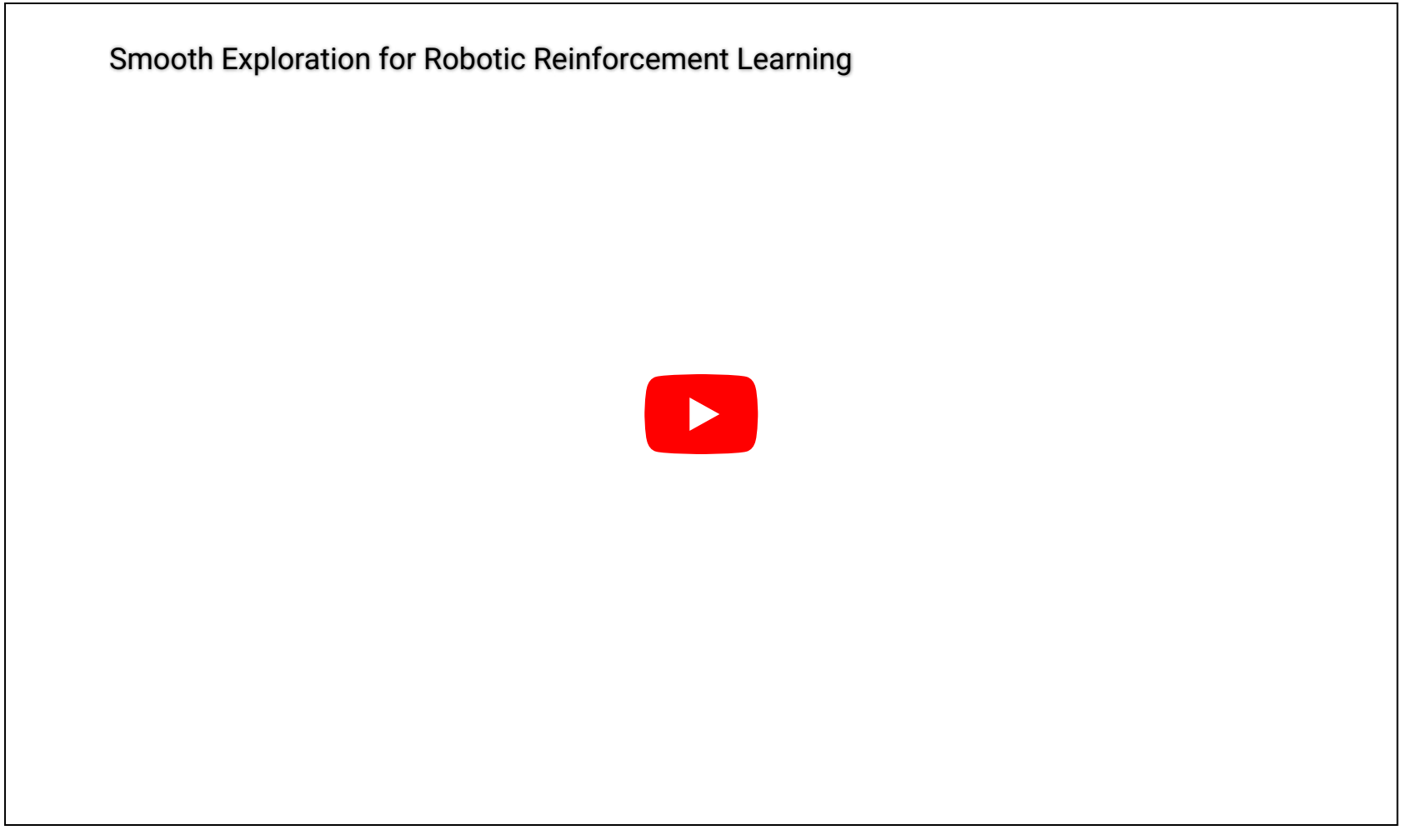


(b) Unstructured Exploration



(c) State Dependent Exploration

SMOOTH EXPLORATION FOR RL



Raffin, Antonin, Jens Kober, and Freek Stulp. "Smooth exploration for robotic reinforcement learning." CoRL. PMLR, 2022.

QUESTIONS?

1. Task design
2. Choosing an algorithm
3. Safety layers
4. **Running the experiments**
5. Troubleshooting

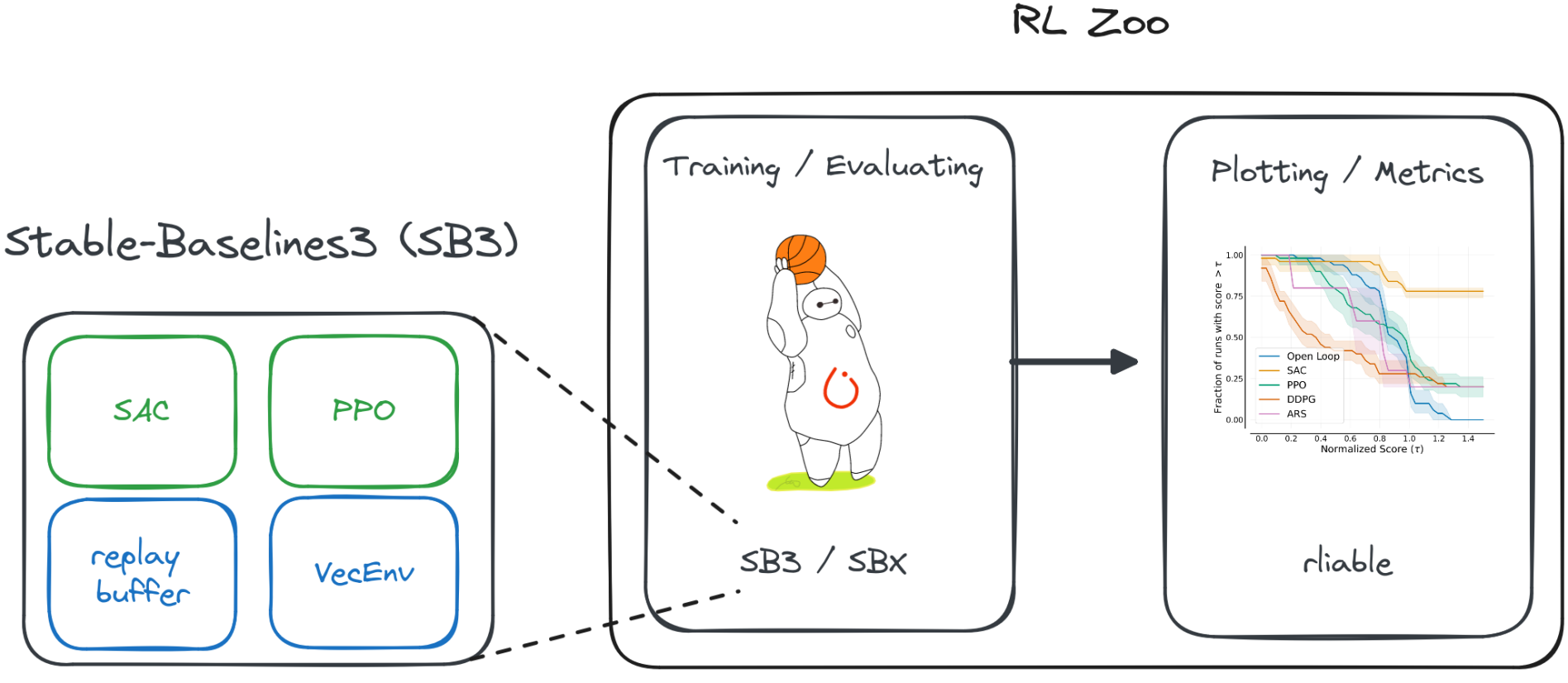
STABLE-BASELINES3 (SB3)

Reliable RL Implementations

```python
import gym
from stable_baselines3 import SAC
# Train an agent using Soft Actor-Critic on Pendulum-v0
env = gym.make("Pendulum-v0")
model = SAC("MlpPolicy", env).learn(total_timesteps=20000)
# Save the model
model.save("sac_pendulum")
# Load the trained model
model = SAC.load("sac_pendulum")
# Start a new episode
obs = env.reset()
# What action to take in state `obs`?
action, _ = model.predict(obs, deterministic=True)
```

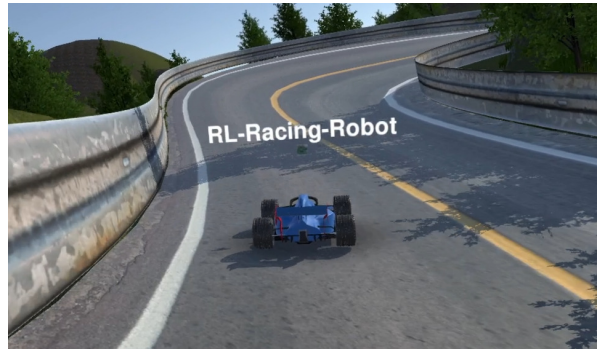
https://github.com/DLR-RM/stable-baselines3

REPRODUCIBLE RELIABLE RL: SB3 + RL ZOO



RL ZOO: REPRODUCIBLE EXPERIMENTS

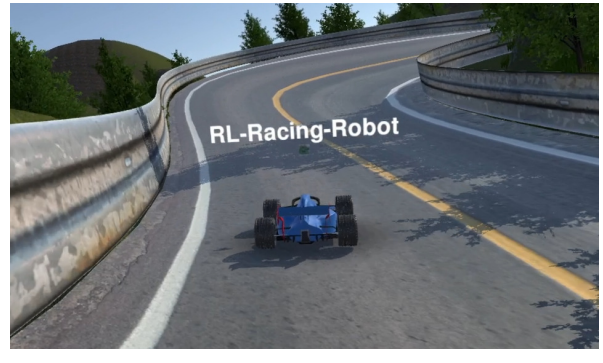
https://github.com/DLR-RM/rl-baselines3-zoo



RL ZOO: REPRODUCIBLE EXPERIMENTS

https://github.com/DLR-RM/rl-baselines3-zoo

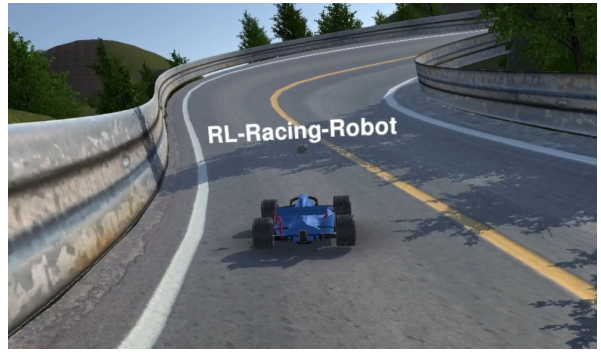
- Training, loading, plotting, hyperparameter optimization



RL ZOO: REPRODUCIBLE EXPERIMENTS

https://github.com/DLR-RM/rl-baselines3-zoo

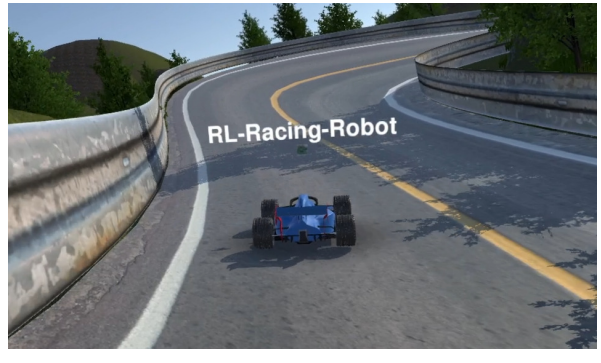
- Training, loading, plotting, hyperparameter optimization
- W&B integration



RL ZOO: REPRODUCIBLE EXPERIMENTS

https://github.com/DLR-RM/rl-baselines3-zoo

- Training, loading, plotting, hyperparameter optimization
- W&B integration
- 200+ trained models with tuned hyperparameters



IN PRACTICE

IN PRACTICE

```
# Train an SAC agent on Pendulum using tuned hyperparameters,
# evaluate the agent every 1k steps and save a checkpoint every 10k steps
# Pass custom hyperparams to the algo/env
python -m rl_zoo3.train --algo sac --env Pendulum-v1 --eval-freq 1000 \
    --save-freq 10000 -params train_freq:2 --env-kwargs g:9.8
```

IN PRACTICE

```
# Train an SAC agent on Pendulum using tuned hyperparameters,
# evaluate the agent every 1k steps and save a checkpoint every 10k steps
# Pass custom hyperparams to the algo/env
python -m rl_zoo3.train --algo sac --env Pendulum-v1 --eval-freq 1000 \
    --save-freq 10000 -params train_freq:2 --env-kwargs g:9.8
```

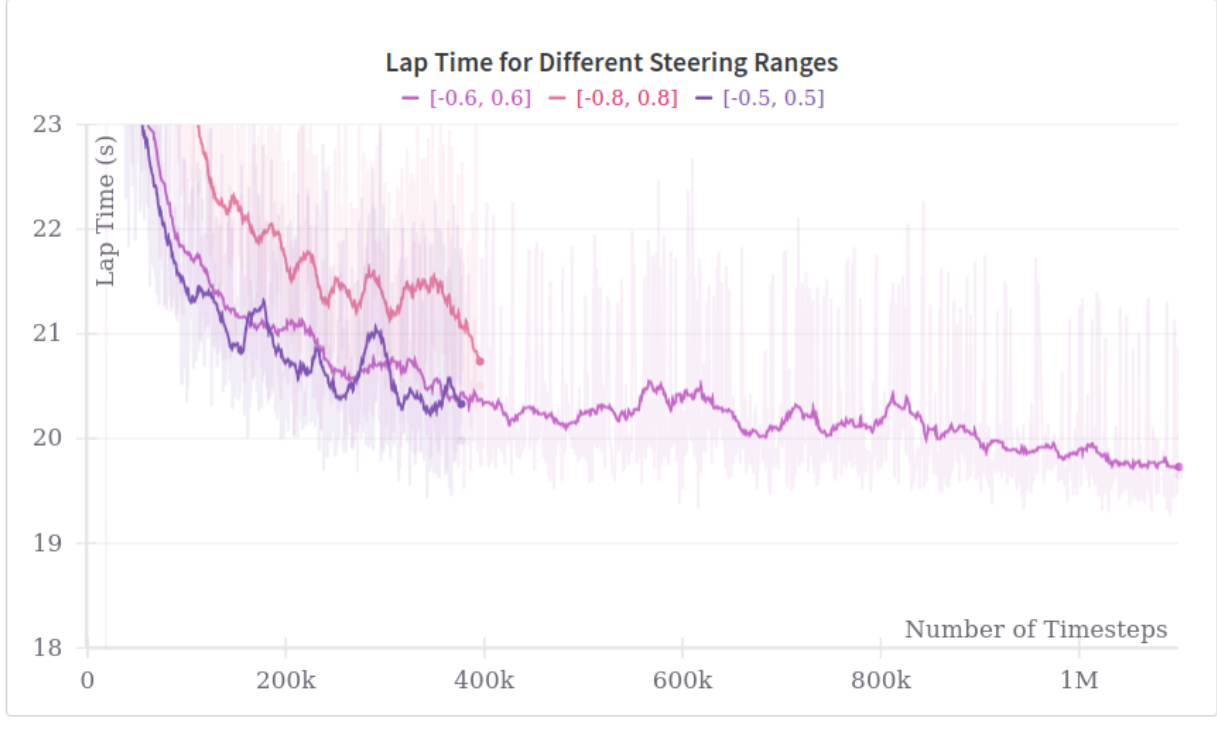
```
sac/
└── Pendulum-v1_1 # One folder per experiment
    ├── 0.monitor.csv # episodic return
    ├── best_model.zip # best model according to evaluation
    ├── evaluations.npz # evaluation results
    ├── Pendulum-v1
    │   ├── args.yml # custom cli arguments
    │   ├── config.yml # hyperparameters
    │   └── vecnormalize.pkl # normalization
    ├── Pendulum-v1.zip # final model
    └── rl_model_10000_steps.zip # checkpoint
```

LEARNING TO RACE IN AN HOUR

DonkeyCar Simulator: Setup and Reinforcement Learning Training (Part 1)

HYPERPARAMETERS STUDY - LEARNING TO RACE

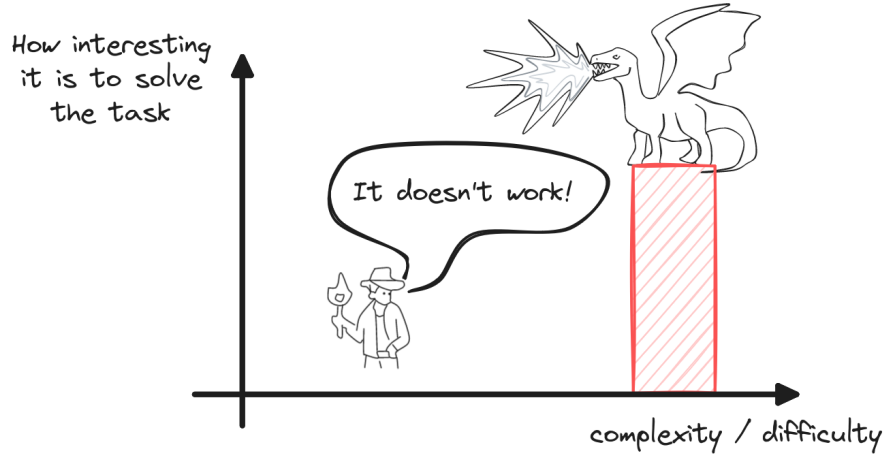
▼ Comparing Different Max Steering



QUESTIONS?

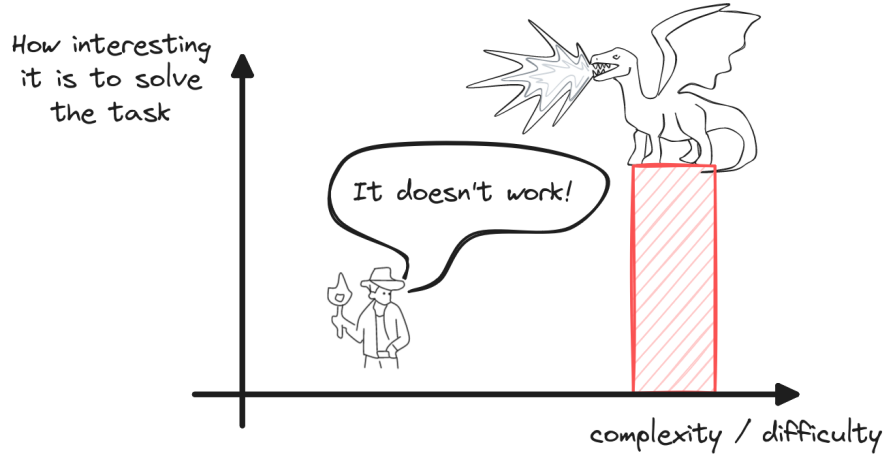
1. Task design
2. Choosing an algorithm
3. Safety layers
4. Running the experiments
5. **Troubleshooting**

IT DOESN'T WORK!



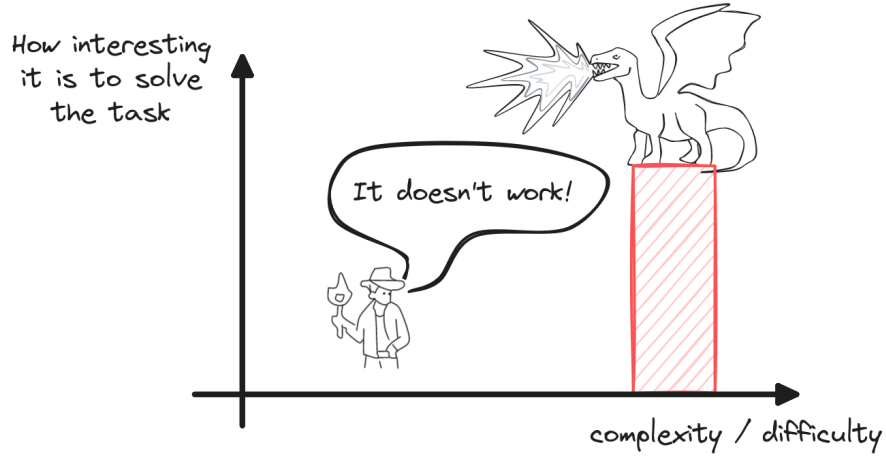
IT DOESN'T WORK!

- Start simple/simplify, iterate quickly



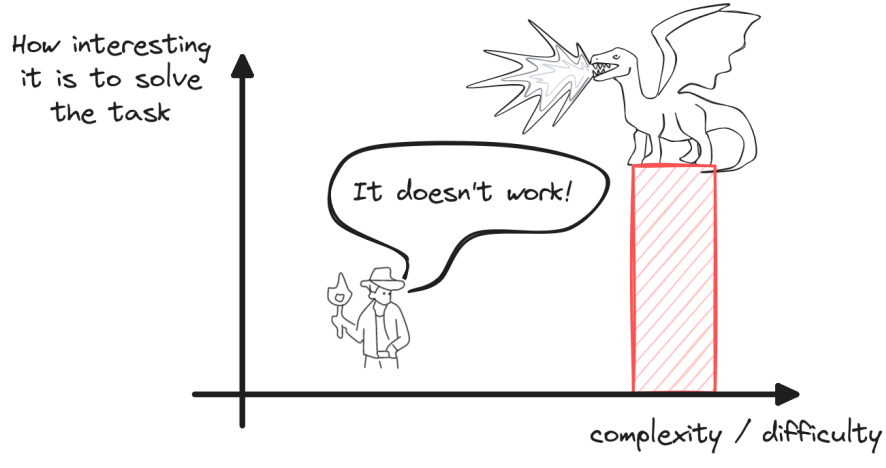
IT DOESN'T WORK!

- Start simple/simplify, iterate quickly
- Did you follow the best practices?



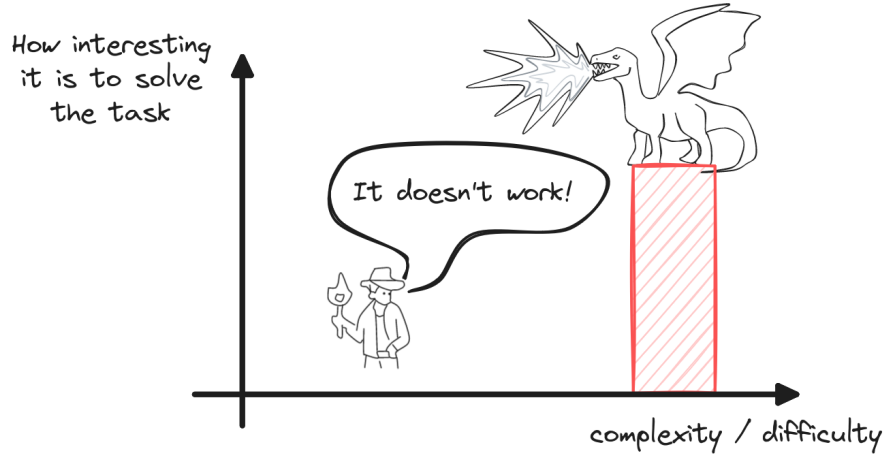
IT DOESN'T WORK!

- Start simple/simplify, iterate quickly
- Did you follow the best practices?
- Use trusted implementations



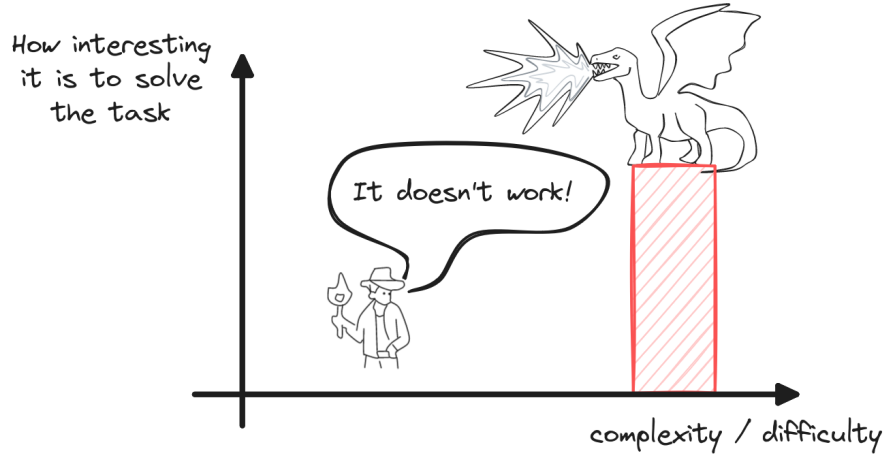
IT DOESN'T WORK!

- Start simple/simplify, iterate quickly
- Did you follow the best practices?
- Use trusted implementations
- Increase budget



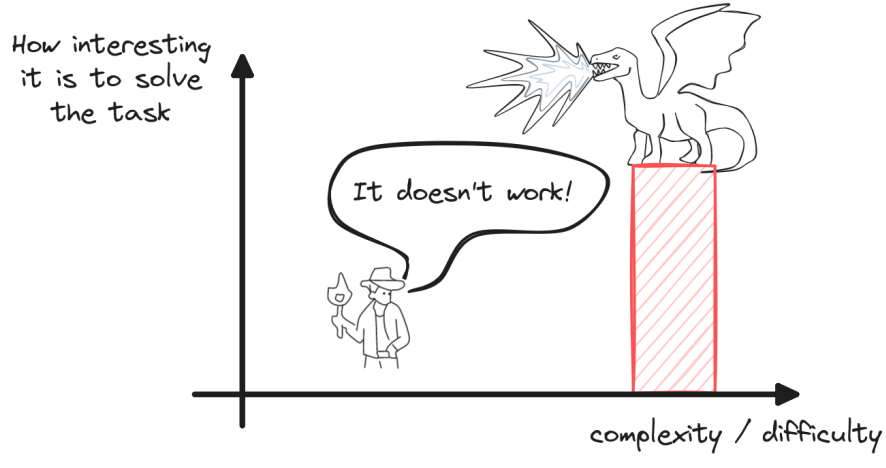
IT DOESN'T WORK!

- Start simple/simplify, iterate quickly
- Did you follow the best practices?
- Use trusted implementations
- Increase budget
- Hyperparameter tuning (Optuna)

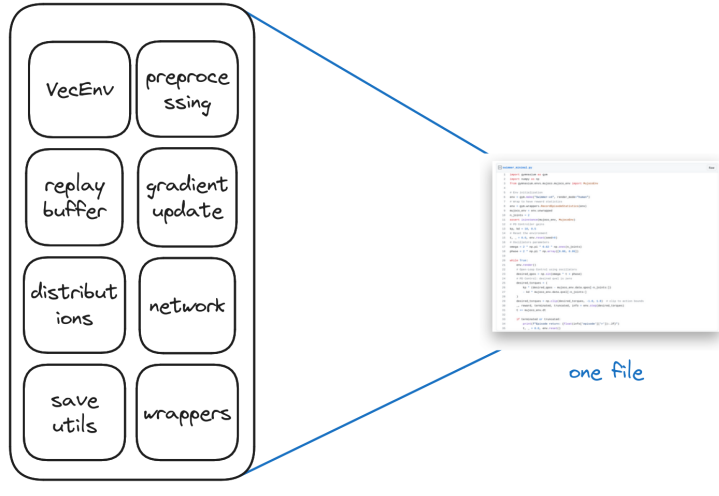


IT DOESN'T WORK!

- Start simple/simplify, iterate quickly
- Did you follow the best practices?
- Use trusted implementations
- Increase budget
- Hyperparameter tuning (Optuna)
- Minimal implementation

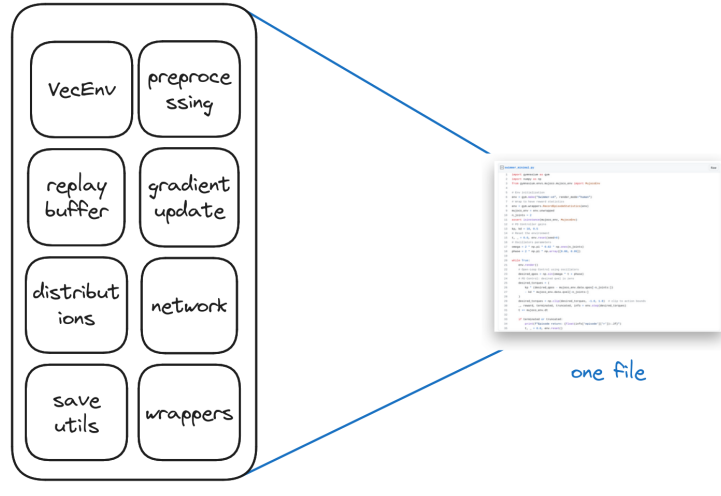


MINIMAL IMPLEMENTATIONS



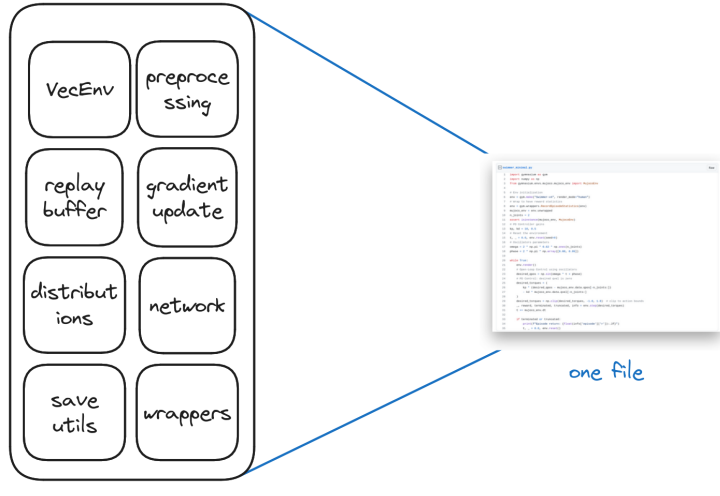
MINIMAL IMPLEMENTATIONS

- Standalone / minimal dependencies



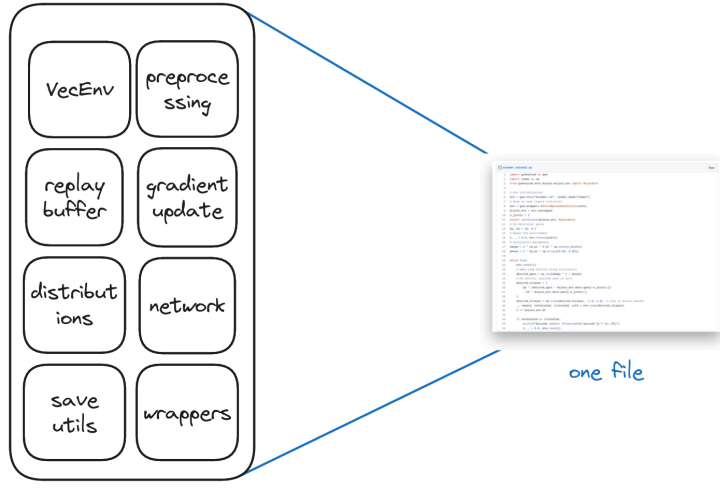
MINIMAL IMPLEMENTATIONS

- Standalone / minimal dependencies
- Reduce complexity



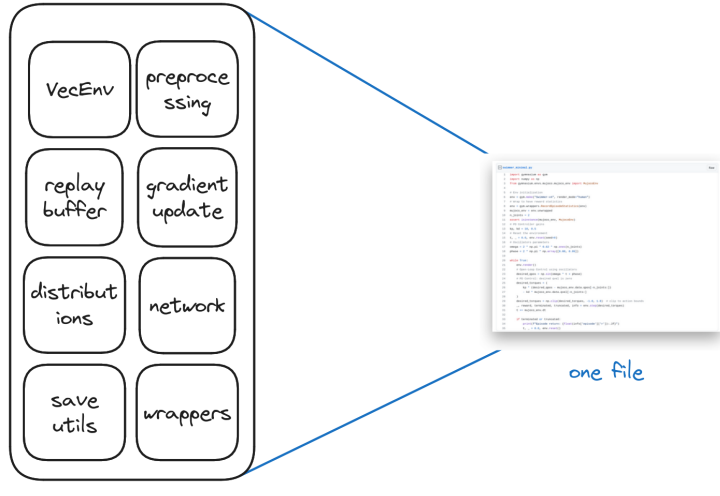
MINIMAL IMPLEMENTATIONS

- Standalone / minimal dependencies
- Reduce complexity
- Easier to share/reproduce



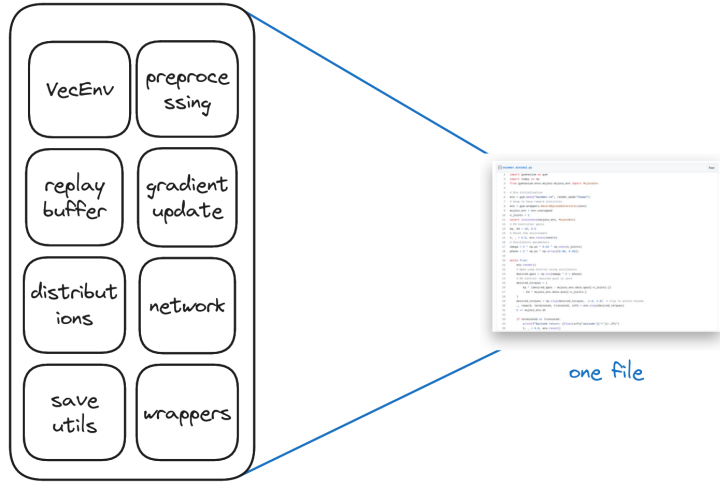
MINIMAL IMPLEMENTATIONS

- Standalone / minimal dependencies
- Reduce complexity
- Easier to share/reproduce
- Perfect for educational purposes



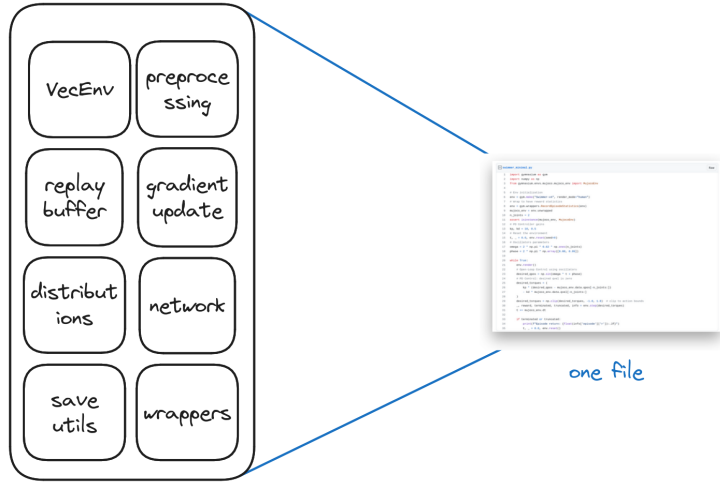
MINIMAL IMPLEMENTATIONS

- Standalone / minimal dependencies
- Reduce complexity
- Easier to share/reproduce
- Perfect for educational purposes
- Find bugs



MINIMAL IMPLEMENTATIONS

- Standalone / minimal dependencies
- Reduce complexity
- Easier to share/reproduce
- Perfect for educational purposes
- Find bugs
- Hard to maintain



EXAMPLE

A Simple Open-Loop Baseline for RL Locomotion Tasks

Raffin et al. "A Simple Open-Loop Baseline for RL Locomotion Tasks" In preparation, CoRL 2024.

35 LINES OF CODE

$$q_i^{\text{des}}(t) = a_i \cdot \sin(\theta_i(t) + \varphi_i) + b_i$$

$$\dot{\theta}_i(t) = \begin{cases} \omega_{\text{swing}} & \text{if } \sin(\theta_i(t) + \varphi_i)) > 0 \\ \omega_{\text{stance}} & \text{otherwise.} \end{cases}$$

35 LINES OF CODE

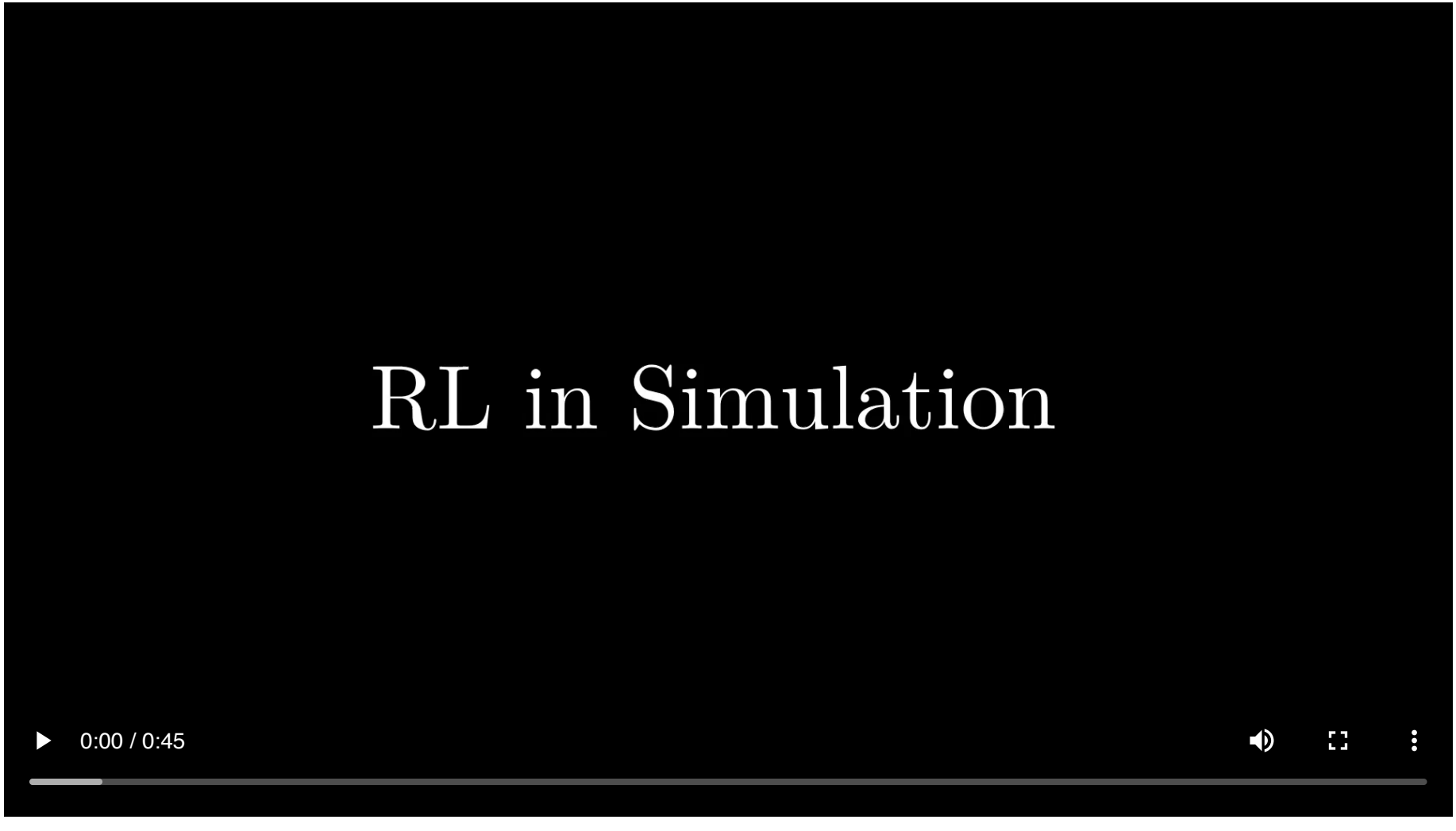
$$q_i^{\text{des}}(t) = a_i \cdot \sin(\theta_i(t) + \varphi_i) + b_i$$

$$\dot{\theta}_i(t) = \begin{cases} \omega_{\text{swing}} & \text{if } \sin(\theta_i(t) + \varphi_i)) > 0 \\ \omega_{\text{stance}} & \text{otherwise.} \end{cases}$$

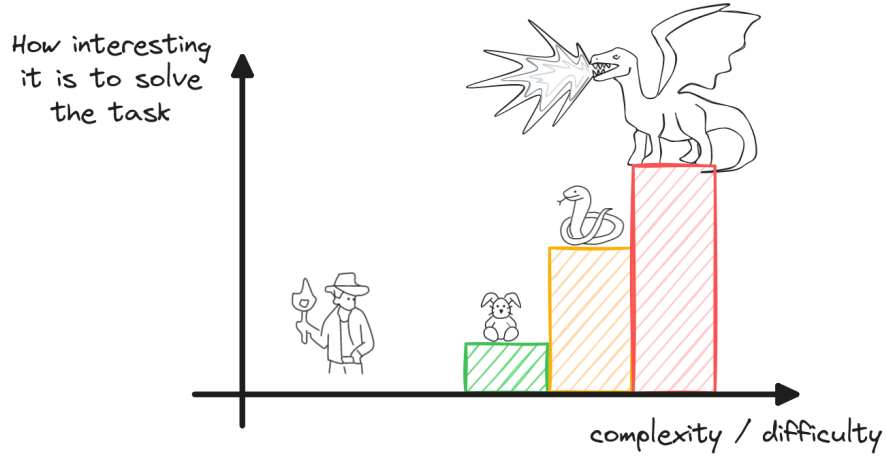
swimmer_minimal.py — Raw

```python
import gymnasium as gym
import numpy as np
from gymnasium.envs.mujoco.mujoco_env import MujocoEnv

# Env initialization
env = gym.make("Swimmer-v4", render_mode="human")
# Wrap to have reward statistics
env = gym.wrappers.RecordEpisodeStatistics(env)
mujoco_env = env.unwrapped
n_joints = 2
assert isinstance(mujoco_env, MujocoEnv)
# PD Controller gains
kp, kd = 10, 0.5
# Reset the environment
t, _ = 0.0, env.reset(seed=0)
# Oscillators parameters
omega = 2 * np.pi * 0.62 * np.ones(n_joints)
phase = 2 * np.pi * np.array([0.00, 0.95])

while True:
    env.render()
    # Open-Loop Control using oscillators
    desired_qpos = np.sin(omega * t + phase)
    # PD Control: desired qvel is zero
    desired_torques = (
        kp * (desired_qpos - mujoco_env.data.qpos[-n_joints:])
        - kd * mujoco_env.data.qvel[-n_joints:]
    )
    desired_torques = np.clip(desired_torques, -1.0, 1.0)  # clip to action bounds
    _, reward, terminated, truncated, info = env.step(desired_torques)
    t += mujoco_env.dt

    if terminated or truncated:
        print(f"Episode return: {float(info['episode']['r']):.2f}")
        t, _ = 0.0, env.reset()
```

COST OF GENERALITY VS PRIOR KNOWLEDGE

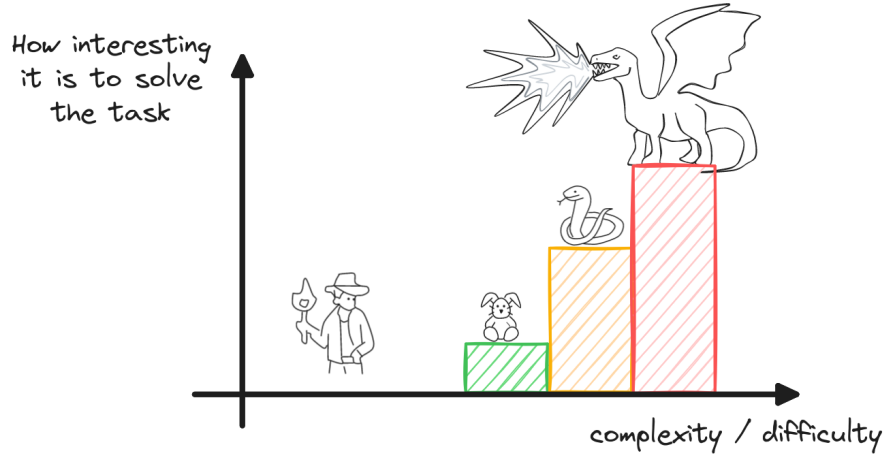


CONCLUSION


How interesting
it is to solve
the task
complexity / difficulty

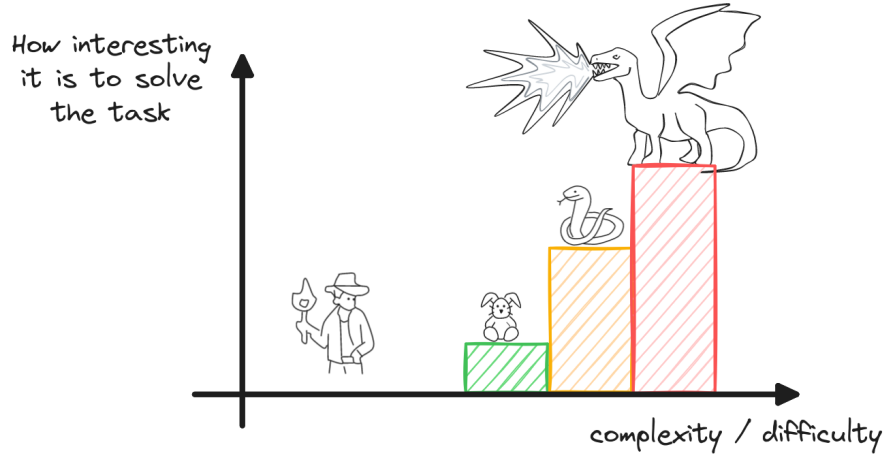
CONCLUSION

- Ingredients for task design



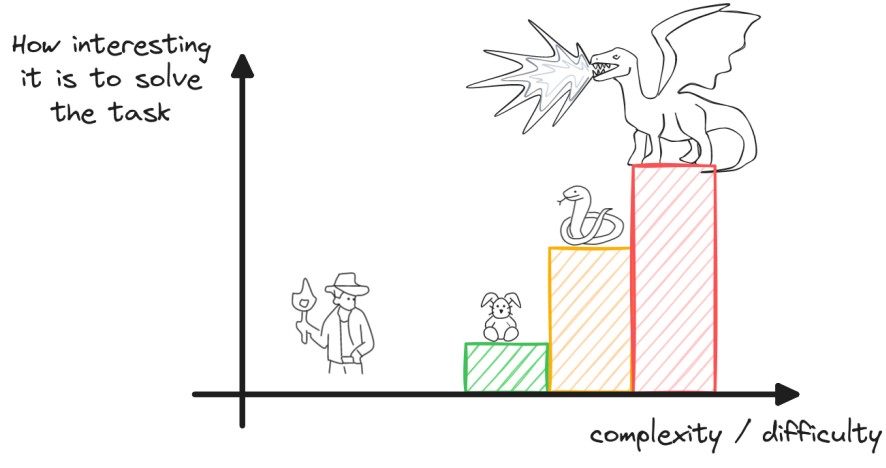
CONCLUSION

- Ingredients for task design
- Safety layers



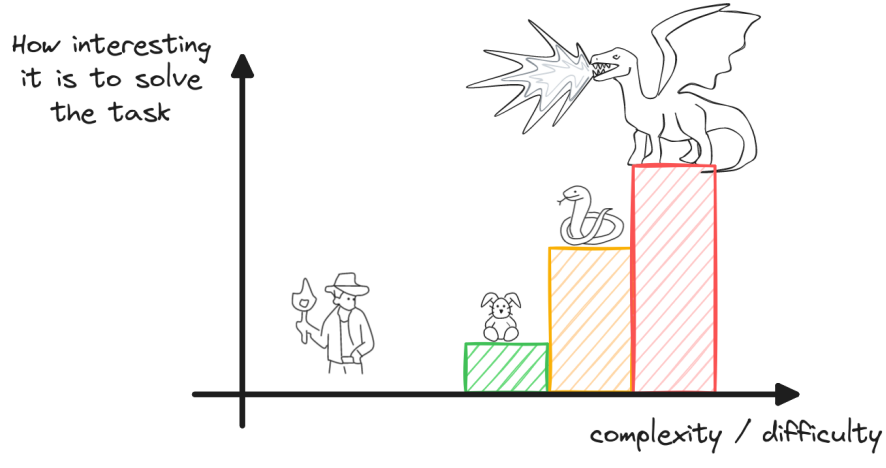
CONCLUSION

- Ingredients for task design
- Safety layers
- Leverage prior knowledge



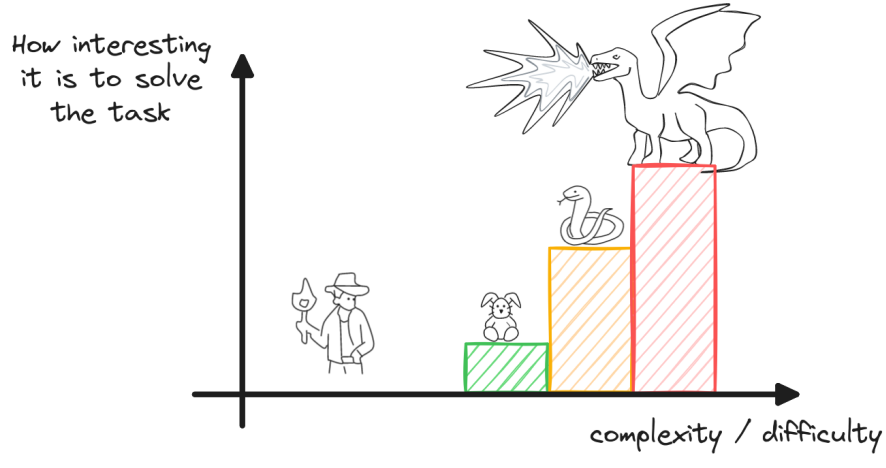
CONCLUSION

- Ingredients for task design
- Safety layers
- Leverage prior knowledge
- Reproducible experiments



CONCLUSION

- Ingredients for task design
- Safety layers
- Leverage prior knowledge
- Reproducible experiments
- Start simple + minimal implementations

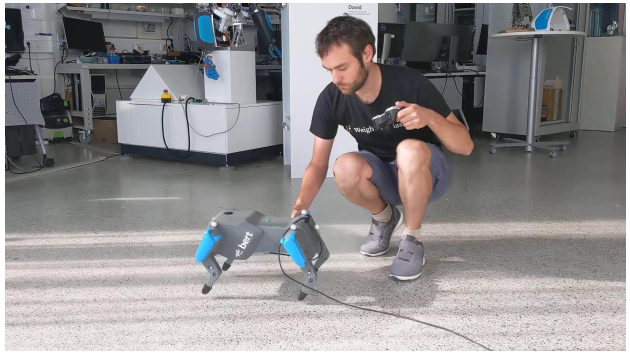


QUESTIONS?

BACKUP SLIDES

WHO AM I?

Stable-Baselines



bert

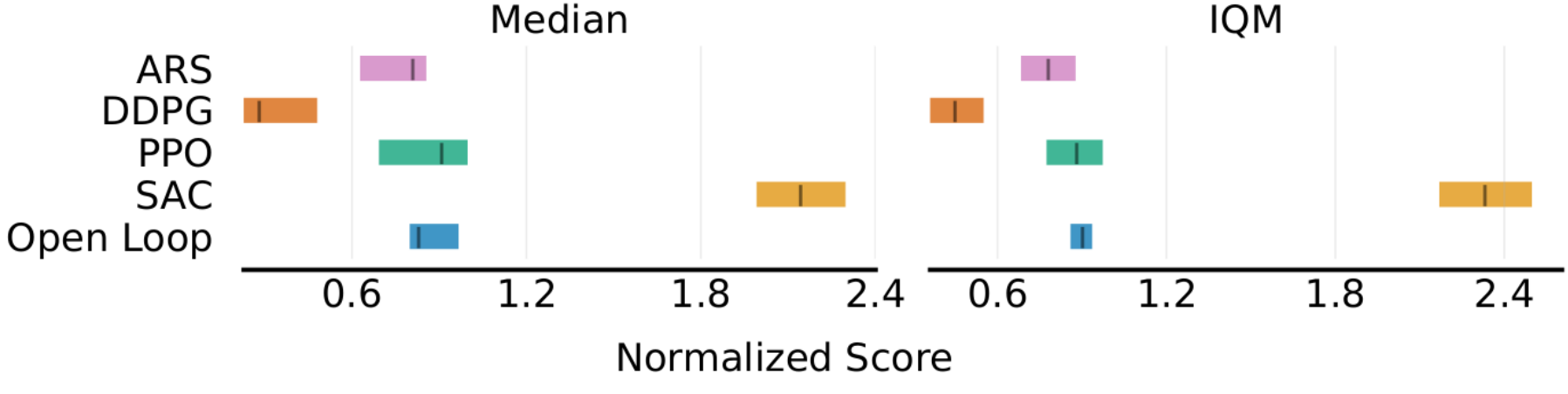
David (aka HASy)

DLR

German Aerospace Center (DLR)

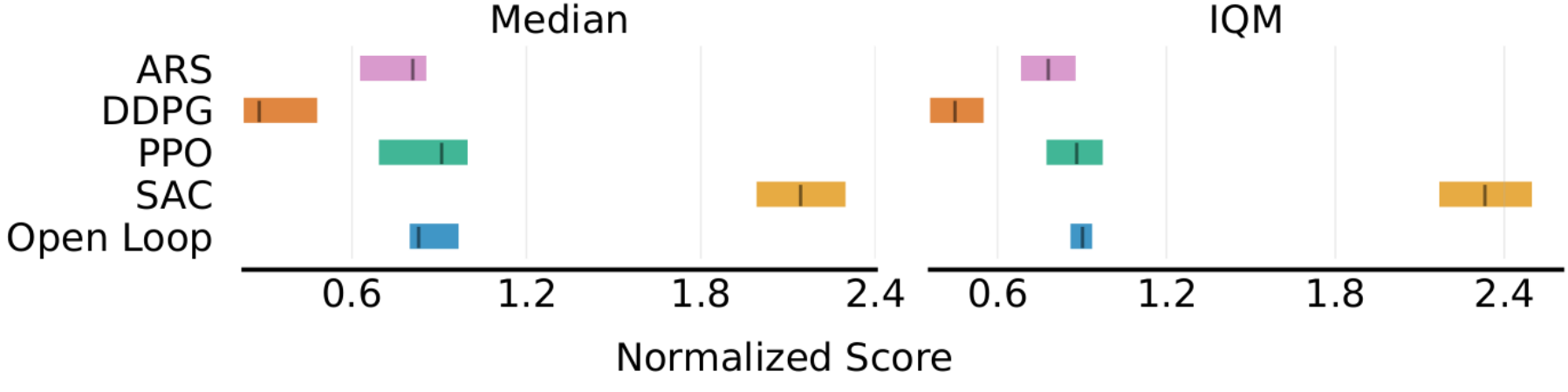
PLOTTING

```
python -m rl_zoo3.cli all_plots -a sac -e HalfCheetah Ant -f logs/ -o sac_results
python -m rl_zoo3.cli plot_from_file -i sac_results.pkl -latex -l SAC --rliable
```



PLOTTING

```
python -m rl_zoo3.cli all_plots -a sac -e HalfCheetah Ant -f logs/ -o sac_results
python -m rl_zoo3.cli plot_from_file -i sac_results.pkl -latex -l SAC --rliable
```



BEST PRACTICES FOR EMPIRICAL RL

BEST PRACTICES FOR EMPIRICAL RL

- Empirical Design in Reinforcement Learning

BEST PRACTICES FOR EMPIRICAL RL

- Empirical Design in Reinforcement Learning
- Rliable: Better Evaluation for Reinforcement Learning

BEST PRACTICES FOR EMPIRICAL RL

- Empirical Design in Reinforcement Learning
- Rliable: Better Evaluation for Reinforcement Learning