DeepLTL: Learning to Efficiently Satisfy Complex LTL Specifications for Multi-Task RL

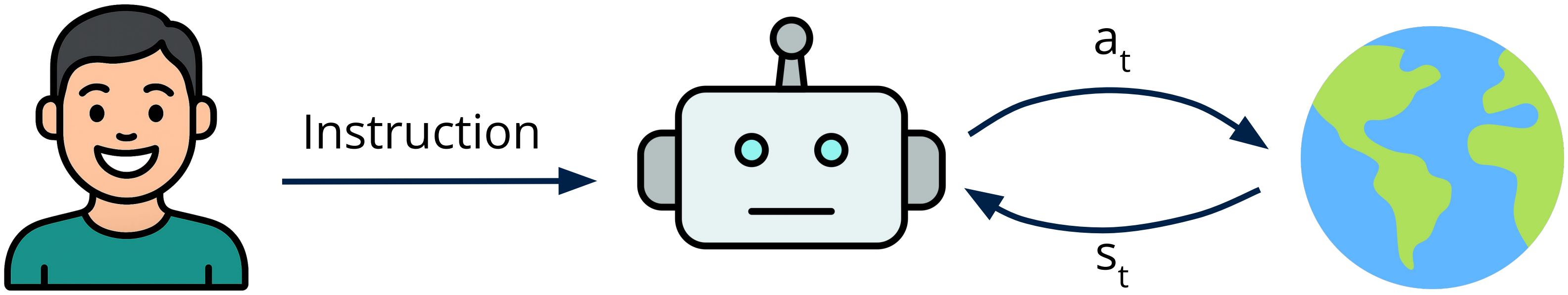
Mathias Jackermeier, Alessandro Abate

ICLR 2025, Singapore

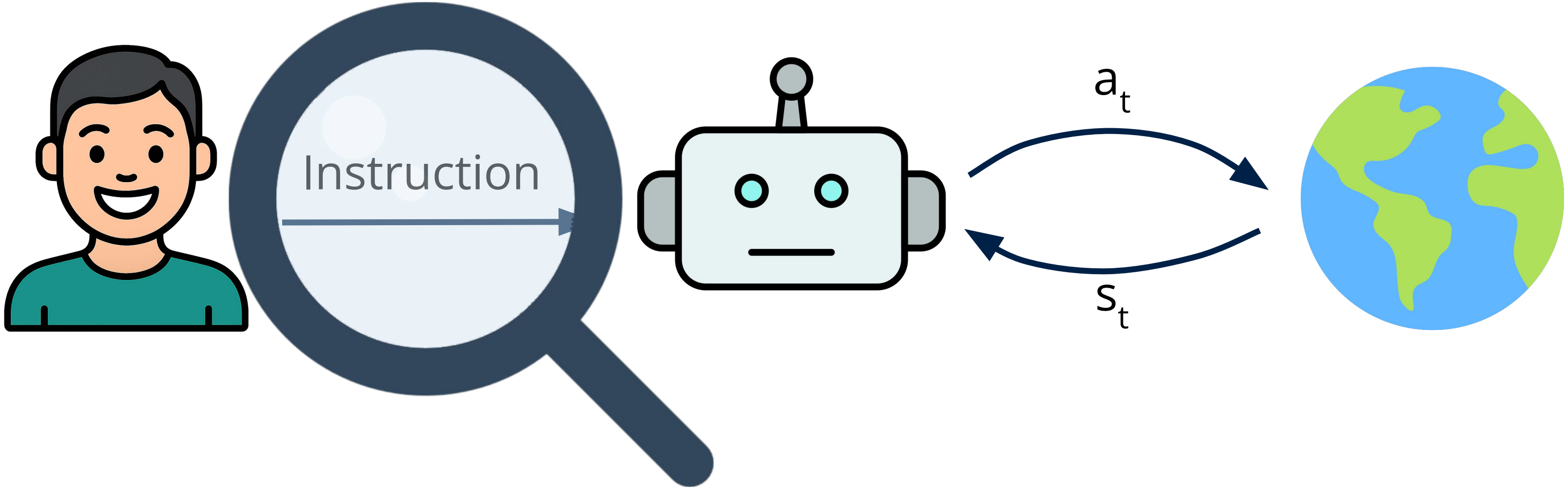
UNIVERSITY OF OXFORD
DEPARTMENT OF COMPUTER SCIENCE

UKRI
Engineering and Physical Sciences Research Council

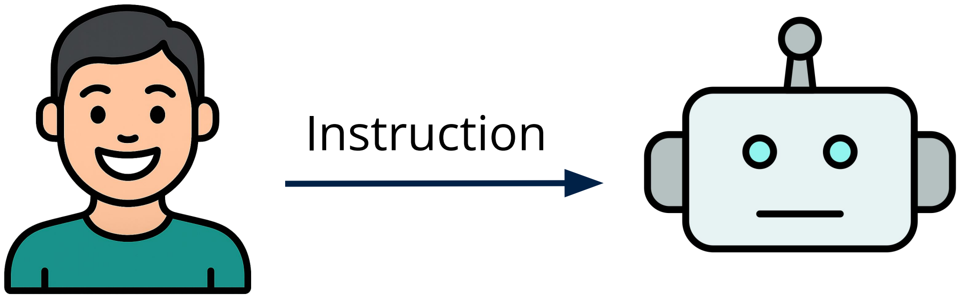
Instruction-following RL agents



Instruction-following RL agents

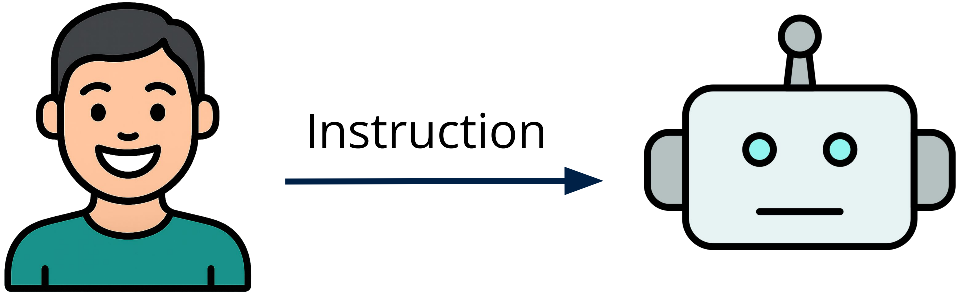


Instruction-following RL agents



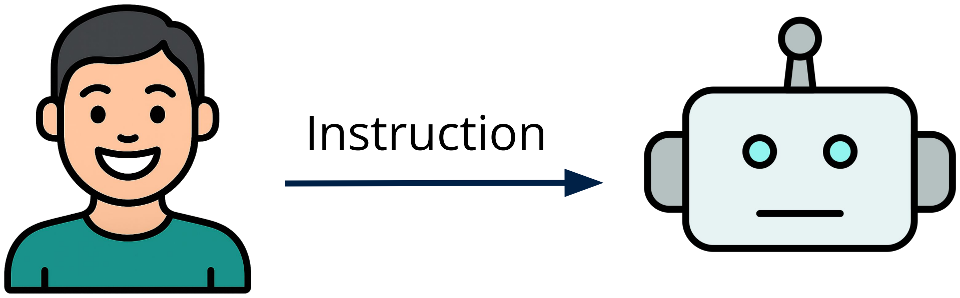
Natural language:
✅ Intuitive
❌ Ambiguous
❌ Difficult to assess

Instruction-following RL agents



Formal specifications:
✅ Precise
✅ Easy to verify
✅ Explicit structure
❌ Difficult to formulate (?)

Instruction-following RL agents



Formal specifications:
✅ Precise
✅ Easy to verify
✅ Explicit structure
❌ Difficult to formulate (?)

→ Well suited when correctness is crucial, e.g. safety-critical settings

Linear temporal logic (LTL)

Example specification:

$$(\neg\mathsf{yellow}\ \mathsf{U}\ \mathsf{purple}) \land \mathsf{G}\,(\mathsf{green} \Rightarrow \mathsf{F}\,\mathsf{blue})$$

Linear temporal logic (LTL)

Example specification:

$$\underbrace{(\neg \mathsf{yellow}\ \mathsf{U}\ \mathsf{purple})} \wedge \mathsf{G}\,(\mathsf{green} \Rightarrow \mathsf{F}\,\mathsf{blue})$$

"Go to the purple zone while avoiding the yellow region,

Linear temporal logic (LTL)

Example specification:

$$(\neg \mathsf{yellow}\ \mathsf{U}\ \mathsf{purple}) \land \mathsf{G}\,(\mathsf{green} \Rightarrow \mathsf{F}\,\mathsf{blue})$$

“Go to the purple zone while avoiding the yellow region, and

Linear temporal logic (LTL)

Example specification:

$$(\neg \mathsf{yellow}\ \mathsf{U}\ \mathsf{purple}) \land \mathsf{G}\,(\mathsf{green} \Rightarrow \mathsf{F}\,\mathsf{blue})$$

"Go to the purple zone while avoiding the yellow region, and always,

Linear temporal logic (LTL)

Example specification:

$$(\neg \mathsf{yellow}\ \mathsf{U}\ \mathsf{purple}) \land \mathsf{G}\,(\mathsf{green} \Rightarrow \mathsf{F}\,\mathsf{blue})$$

"Go to the purple zone while avoiding the yellow region, and always, if you visit green you eventually have to go to blue."

Linear temporal logic (LTL)

Example specification:

$$(\neg \mathsf{yellow}\ \mathsf{U}\ \mathsf{purple}) \wedge \mathsf{G}\,(\mathsf{green} \Rightarrow \mathsf{F}\,\mathsf{blue})$$

"Go to the purple zone while avoiding the yellow region, and always, if you visit green you eventually have to go to blue."

Linear temporal logic (LTL)

Example specification:

$$(\neg \mathsf{yellow}\ \mathsf{U}\ \mathsf{purple}) \wedge \mathsf{G}\,(\mathsf{green} \Rightarrow \mathsf{F}\,\mathsf{blue})$$

"Go to the purple zone while avoiding the yellow region, and always, if you visit green you eventually have to go to blue."

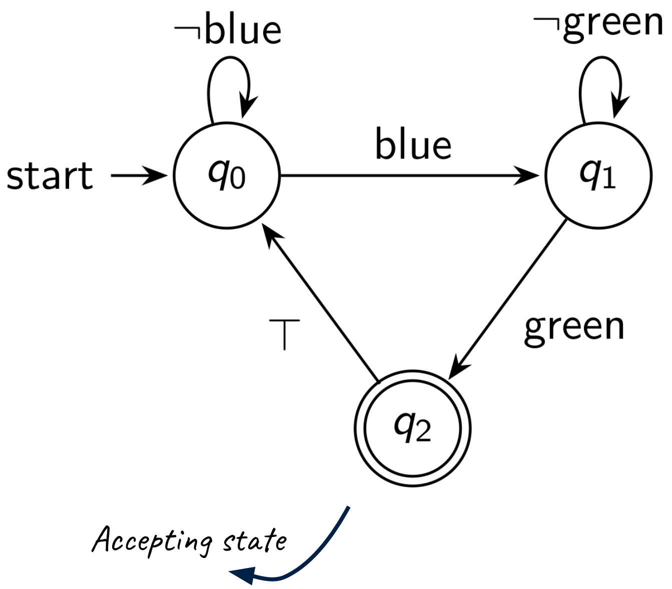
✅ Compositional ✅ Temporally extended ✅ Infinite horizon ✅ Safety constraints

How can we train a **multi-task** policy to **zero-shot** execute **arbitrary** LTL specifications?

From LTL specifications to automata

Any LTL specification can be converted to an equivalent (Büchi) **automaton**:

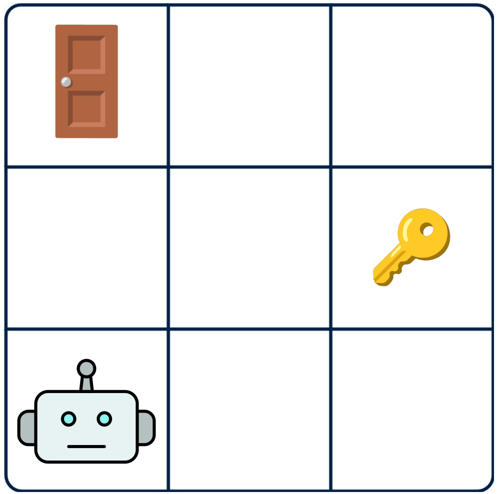
$$\mathsf{G}\,\mathsf{F}\,\mathsf{blue} \wedge \mathsf{G}\,\mathsf{F}\,\mathsf{green} \longrightarrow$$

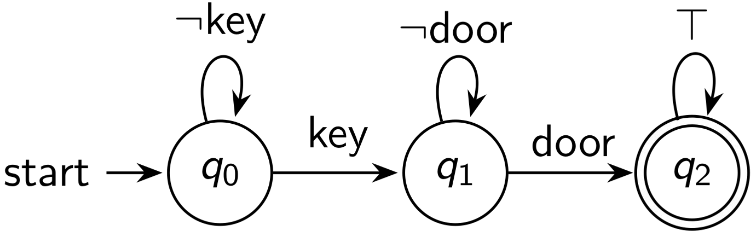


Product MDP

Keeping track of the automaton state allows us to learn a **Markovian** policy

$$\pi : \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$$



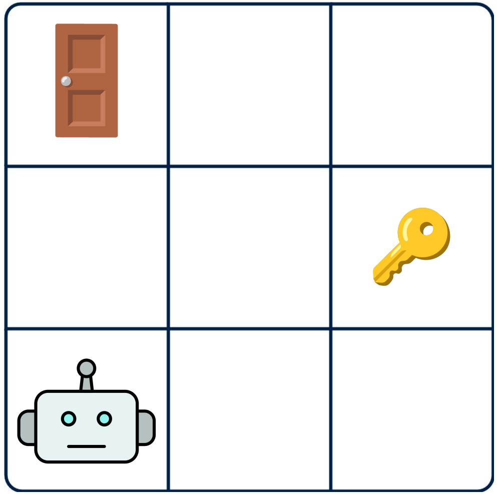


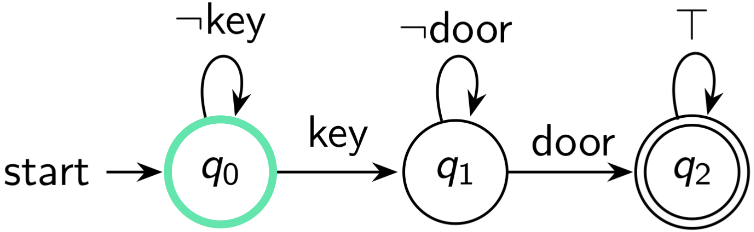
F (key ∧ F door)

Product MDP

Keeping track of the automaton state allows us to learn a **Markovian** policy

$$\pi : \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$$



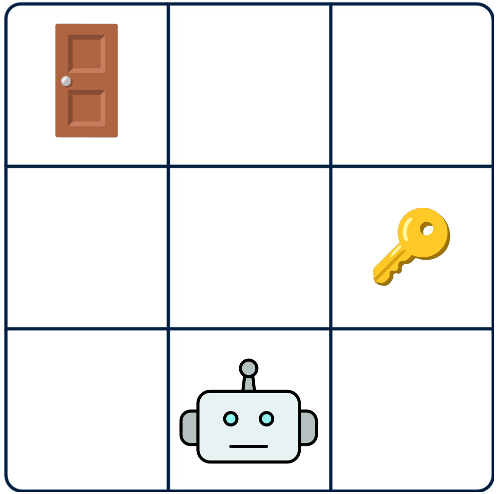


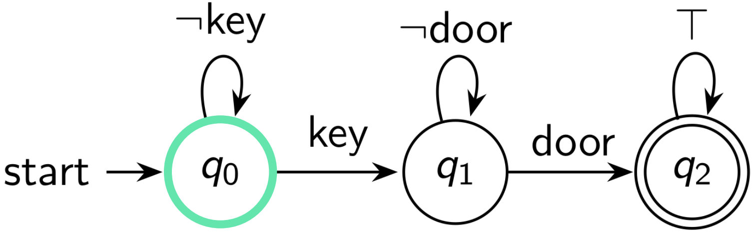
F (key ∧ F door)

Product MDP

Keeping track of the automaton state allows us to learn a **Markovian** policy

$$\pi : \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$$



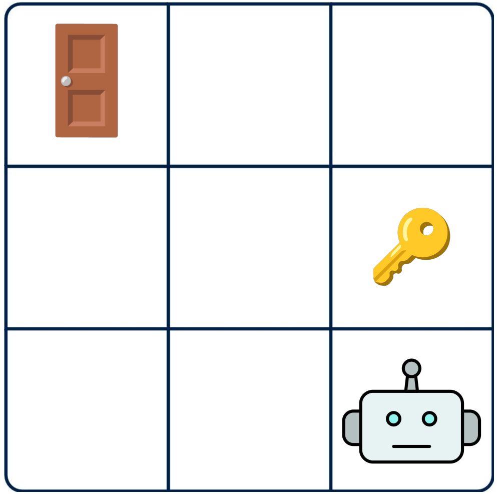


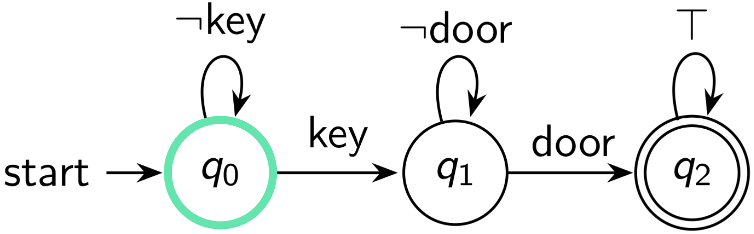
F (key ∧ F door)

Product MDP

Keeping track of the automaton state allows us to learn a **Markovian** policy

$$\pi : \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$$



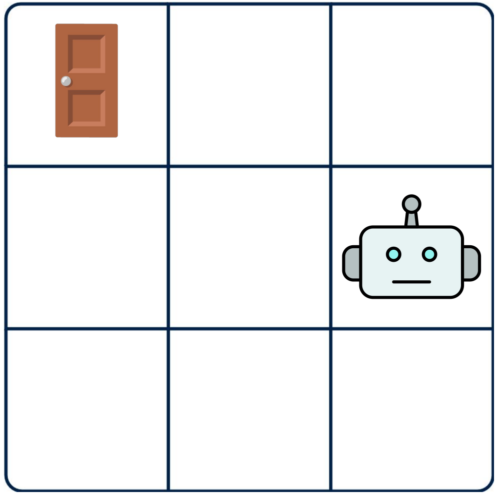


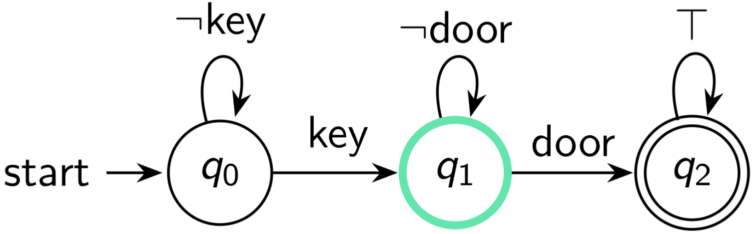
F (key ∧ F door)

Product MDP

Keeping track of the automaton state allows us to learn a **Markovian** policy

$$\pi : \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$$



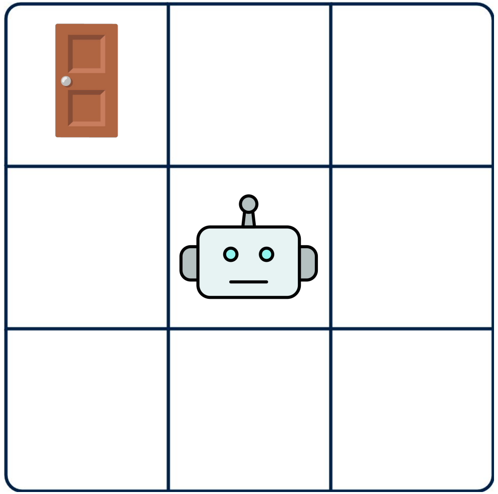


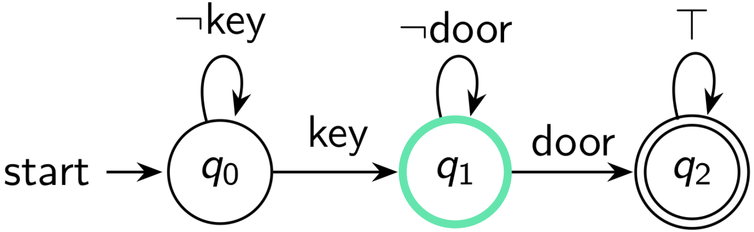
$\mathsf{F}\,(\mathsf{key} \wedge \mathsf{F}\,\mathsf{door})$

Product MDP

Keeping track of the automaton state allows us to learn a **Markovian** policy

$$\pi : \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$$



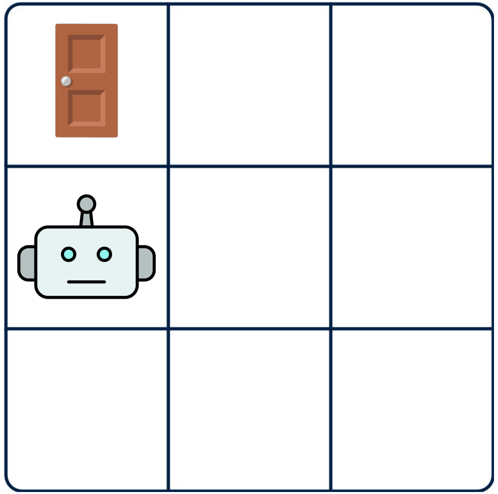


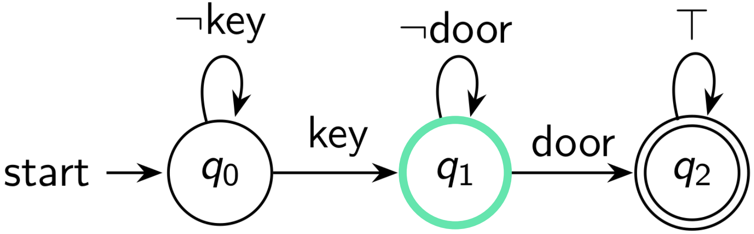
$\mathsf{F}\,(\mathsf{key} \wedge \mathsf{F}\,\mathsf{door})$

Product MDP

Keeping track of the automaton state allows us to learn a **Markovian** policy

$$\pi : \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$$



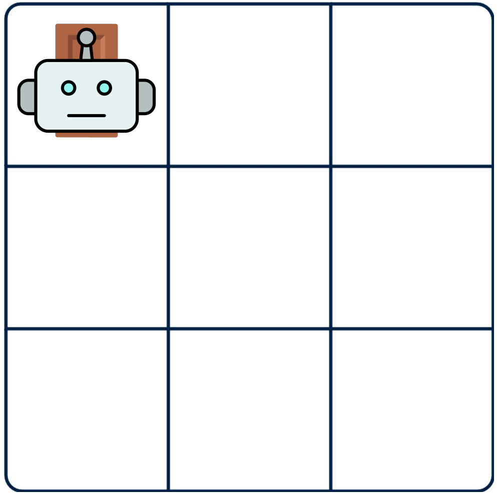


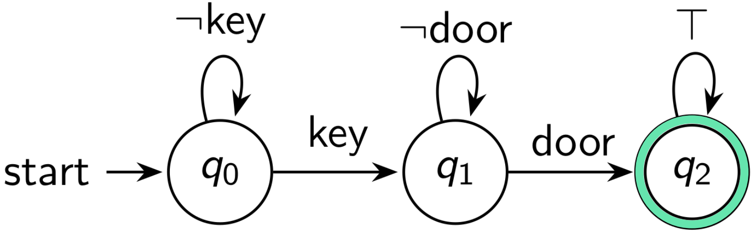
$\mathsf{F}\,(\mathsf{key} \land \mathsf{F}\,\mathsf{door})$

Product MDP

Keeping track of the automaton state allows us to learn a **Markovian** policy

$$\pi : \mathcal{S} \times \mathcal{Q} \to \Delta(\mathcal{A})$$

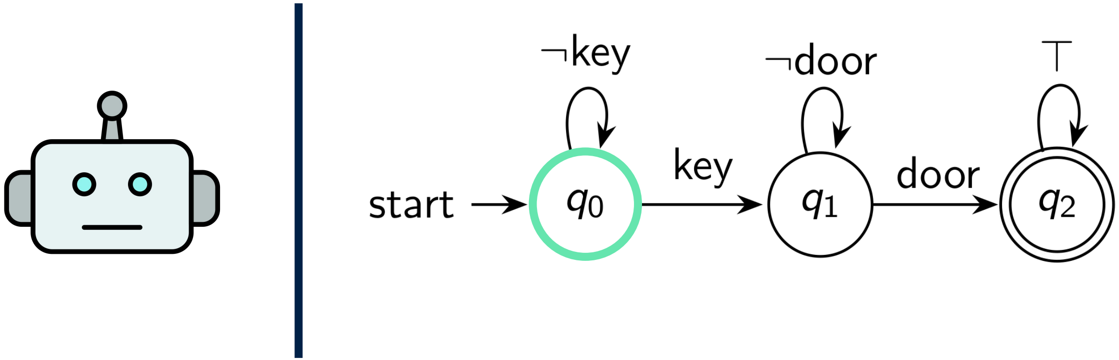




$\mathsf{F}\,(\mathsf{key} \wedge \mathsf{F}\,\mathsf{door})$

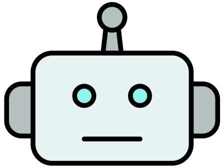
From single-task to multi-task

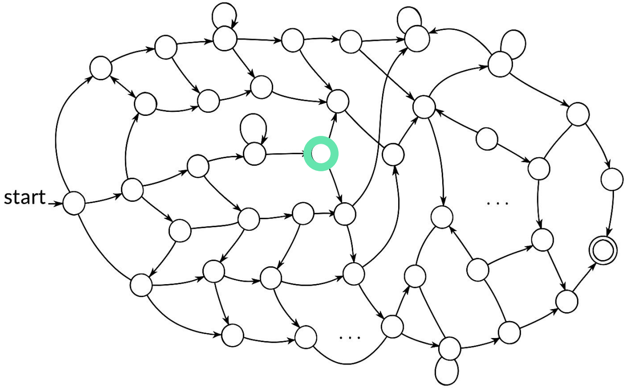
In a multi-task setting, we do not know the automaton beforehand



From single-task to multi-task

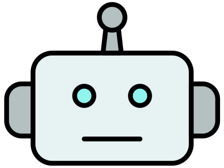
In a multi-task setting, we do not know the automaton beforehand

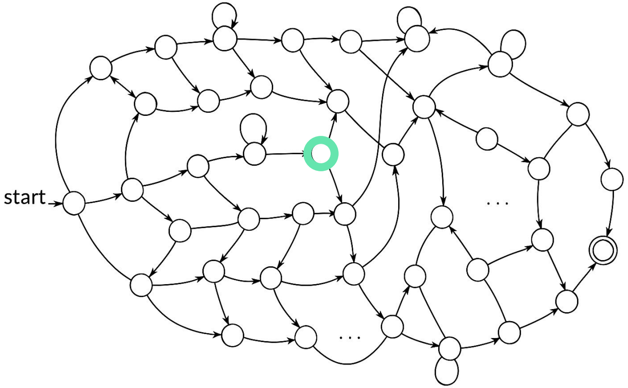




From single-task to multi-task

In a multi-task setting, we do not know the automaton beforehand

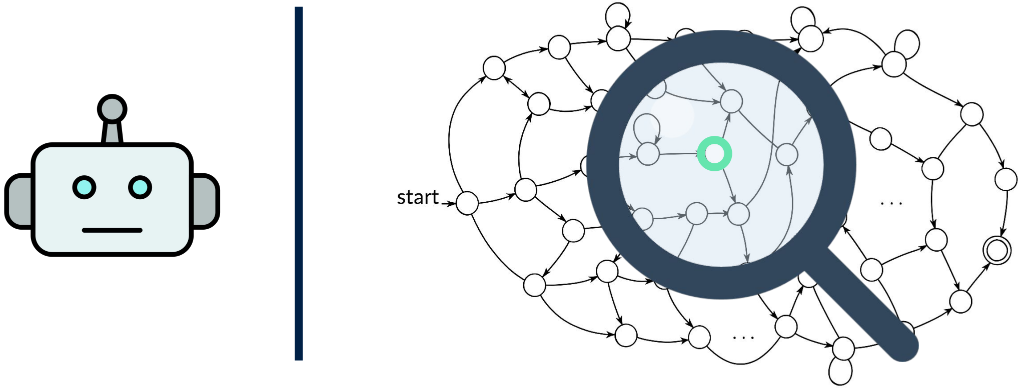




What is a **general representation** of the automaton state that can be used to condition the policy?

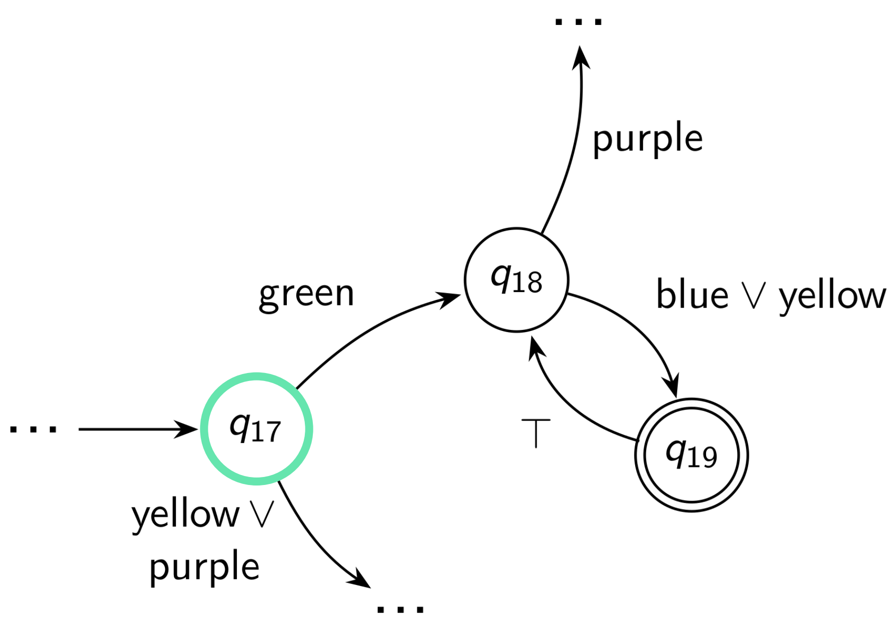
From single-task to multi-task

In a multi-task setting, we do not know the automaton beforehand

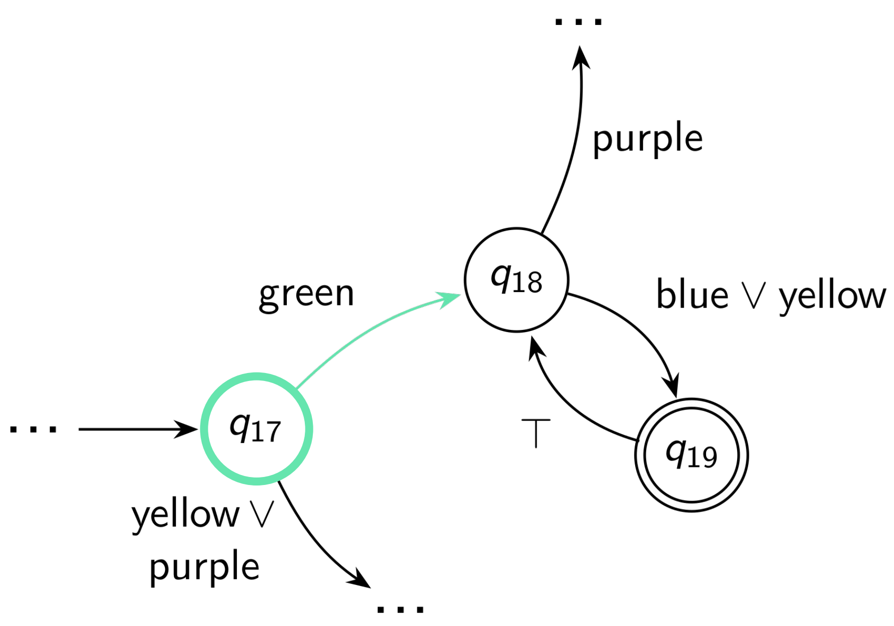


What is a **general representation** of the automaton state that can be used to condition the policy?

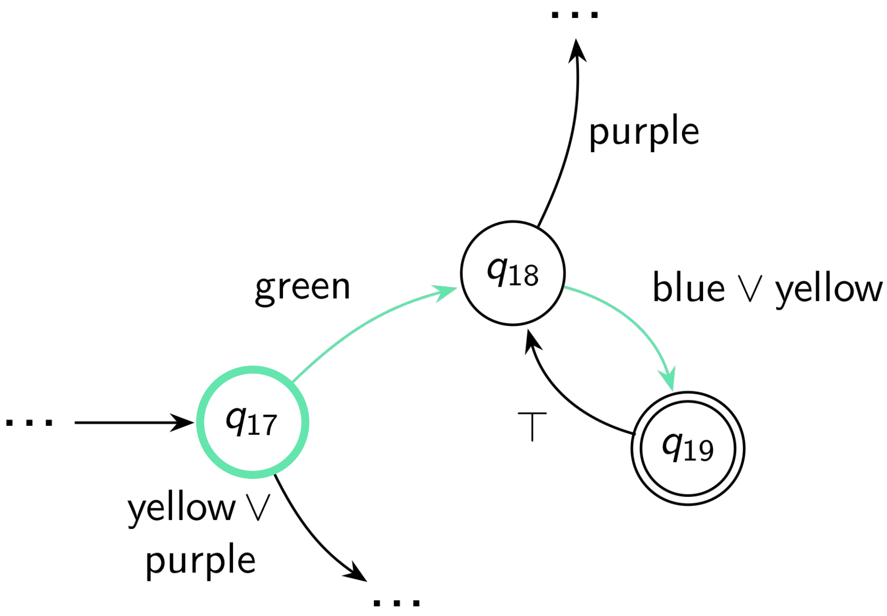
Reach-avoid sequences



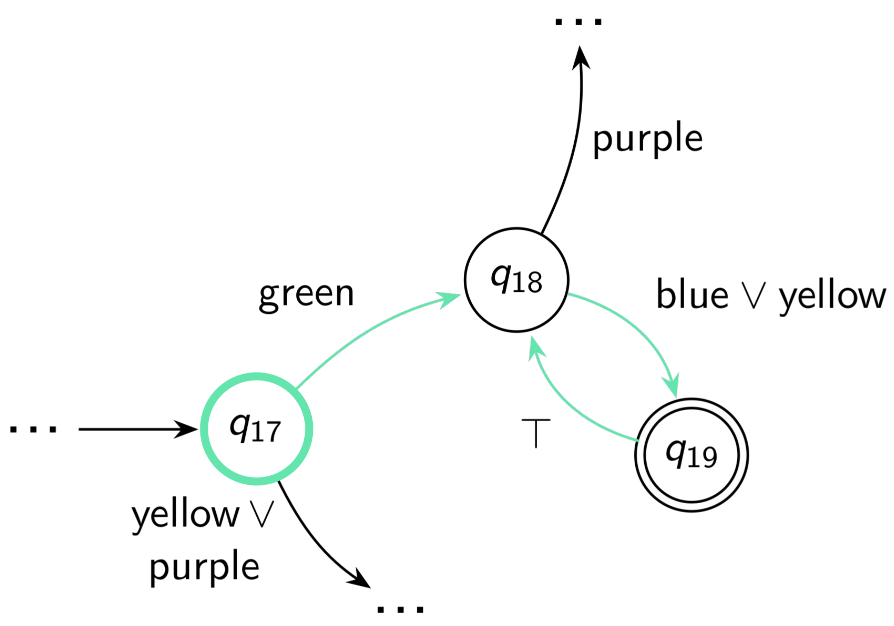
Reach-avoid sequences



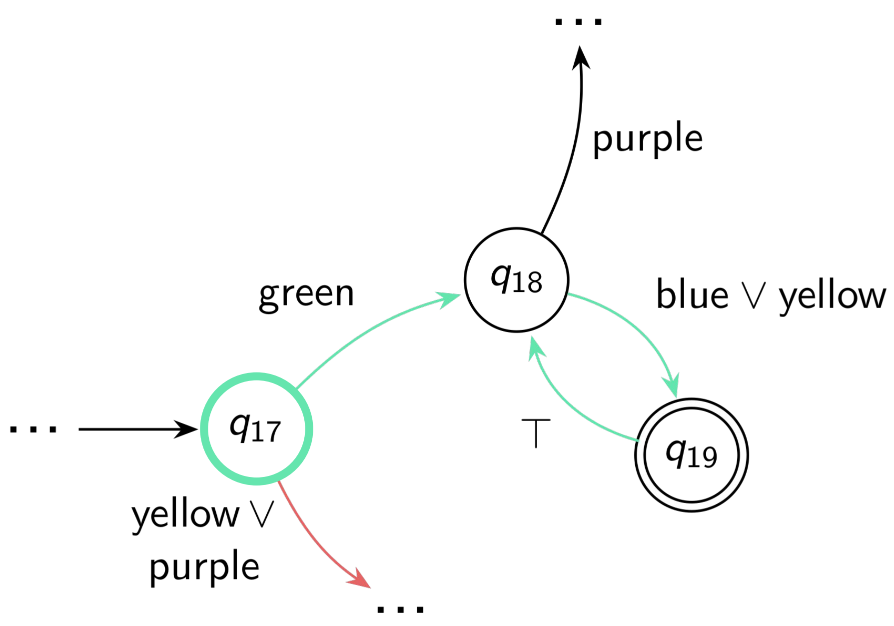
Reach-avoid sequences



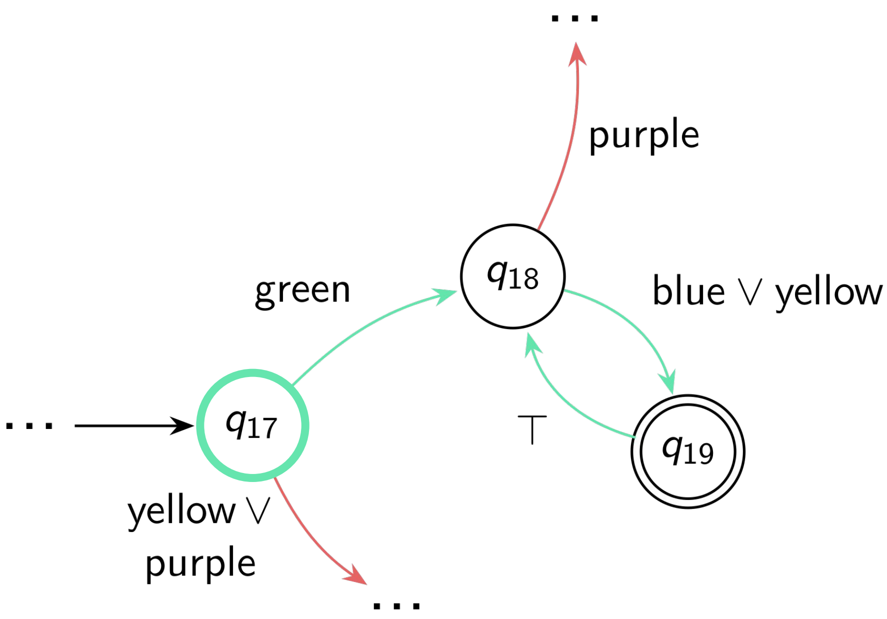
Reach-avoid sequences



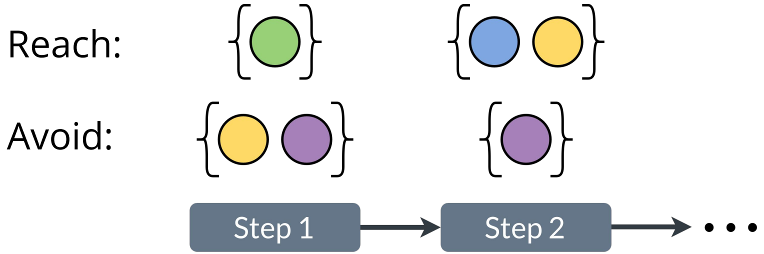
Reach-avoid sequences



Reach-avoid sequences

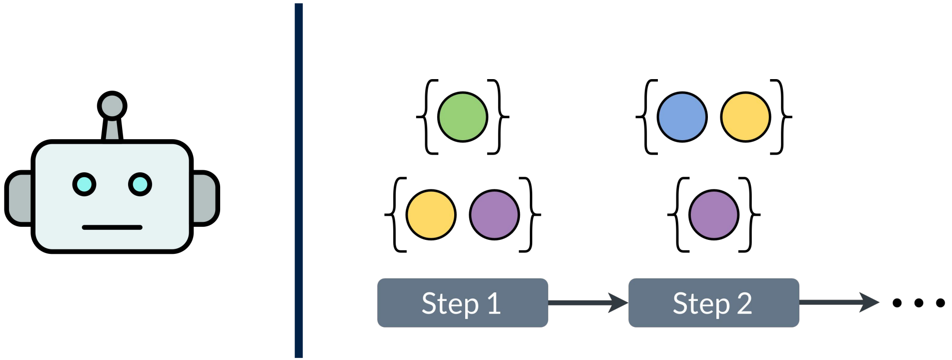


Reach-avoid sequences



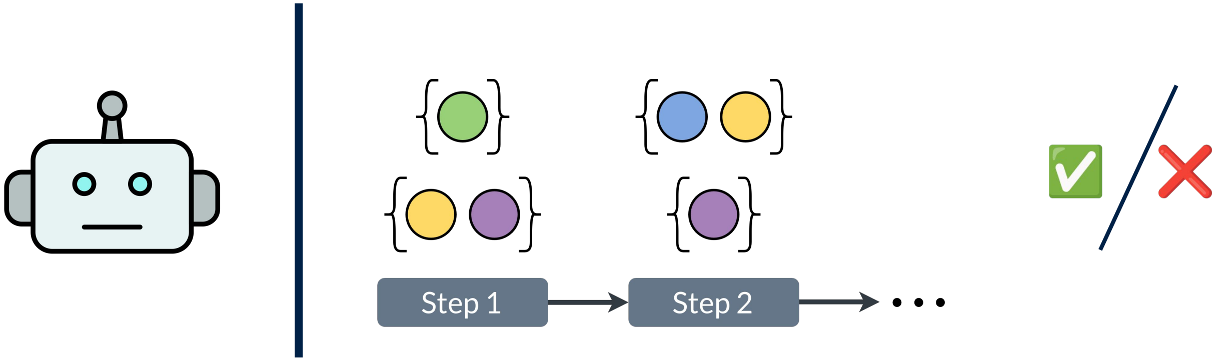
Training a general policy

We use goal-conditioned RL to train a general policy:



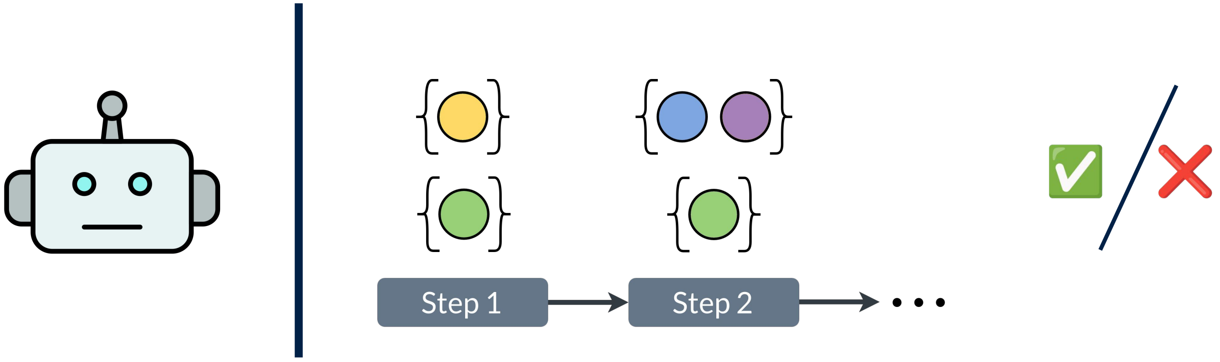
Training a general policy

We use goal-conditioned RL to train a general policy:



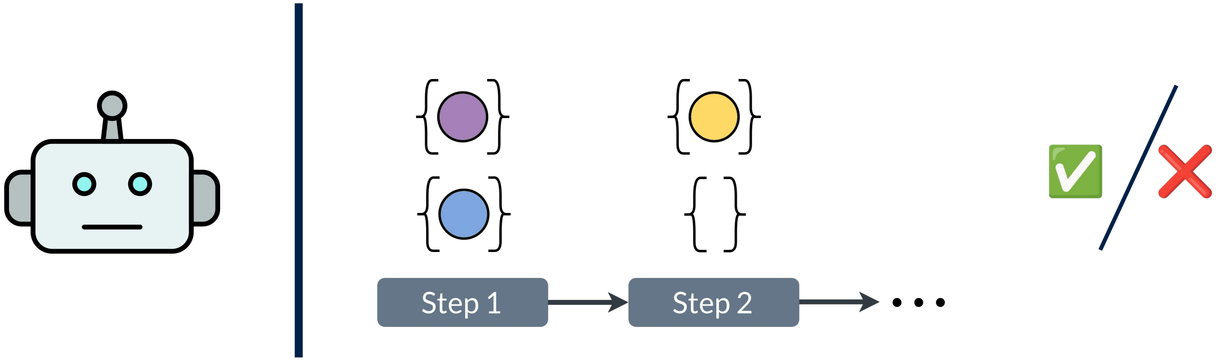
Training a general policy

We use goal-conditioned RL to train a general policy:



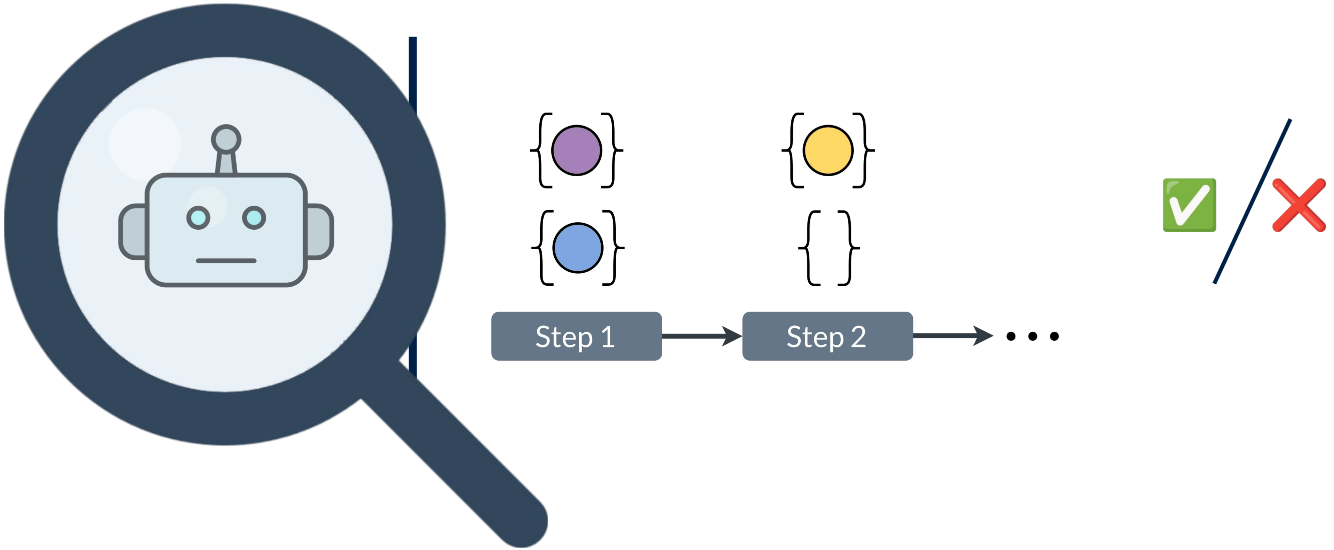
Training a general policy

We use goal-conditioned RL to train a general policy:

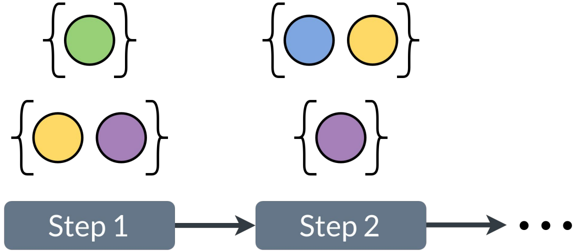


Training a general policy

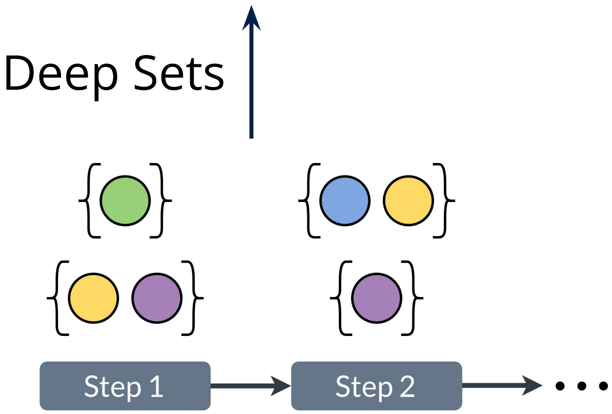
We use goal-conditioned RL to train a general policy:



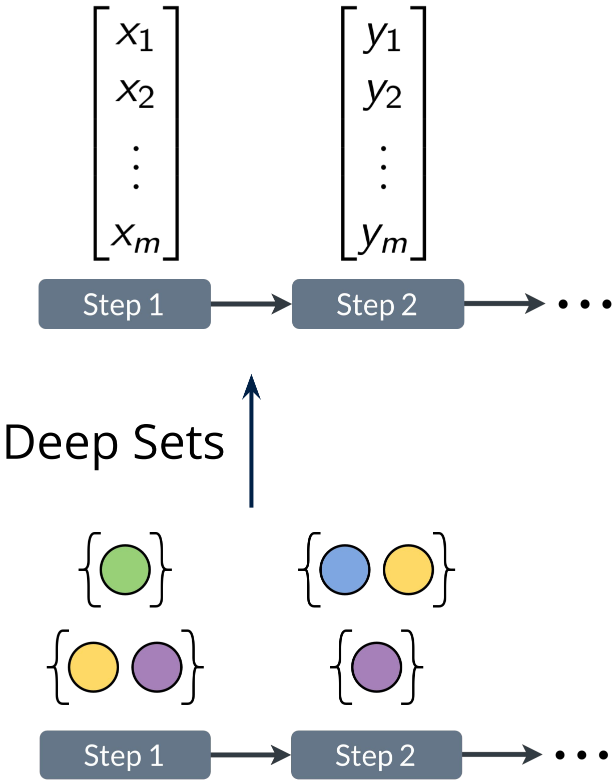
Model architecture



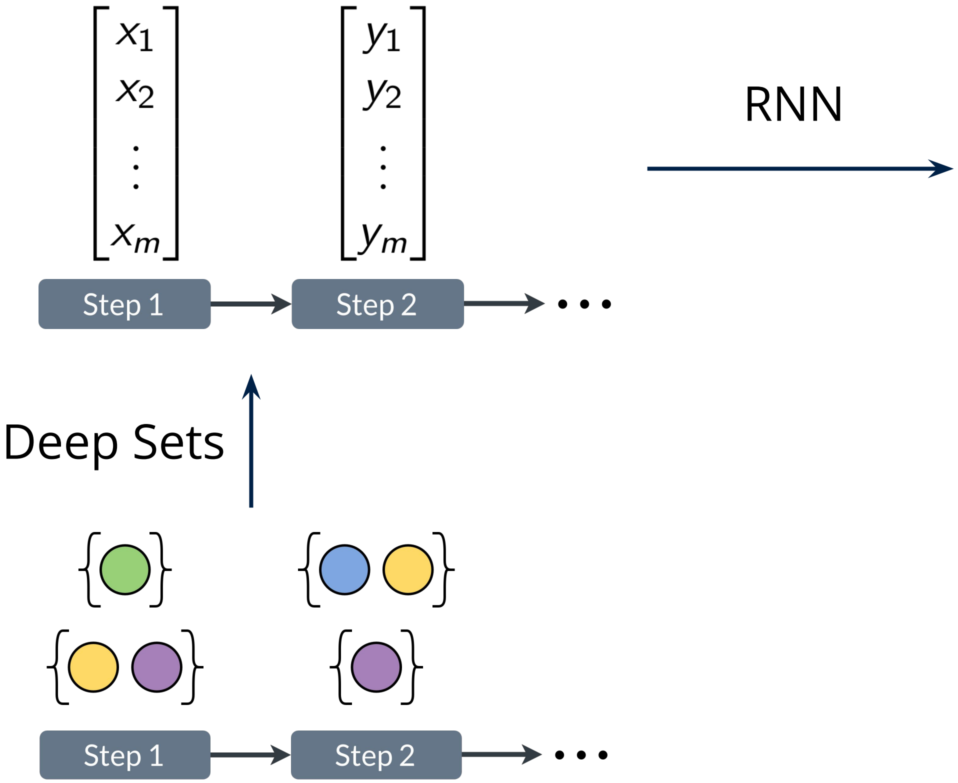
Model architecture



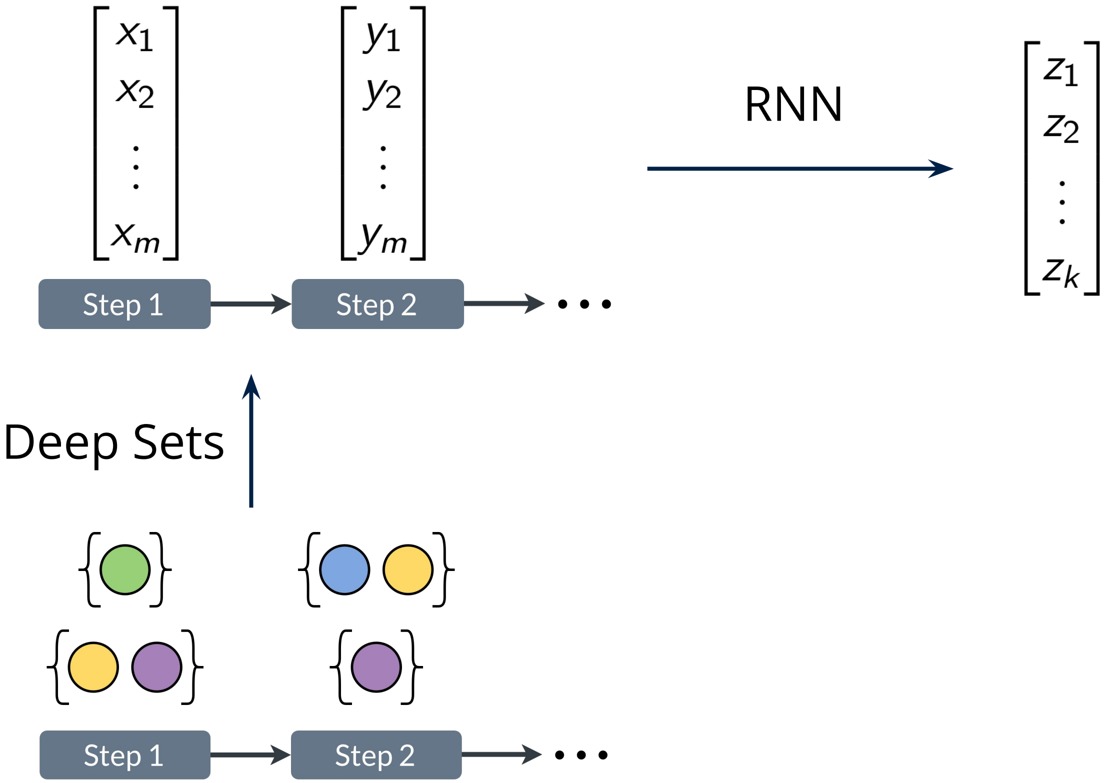
Model architecture



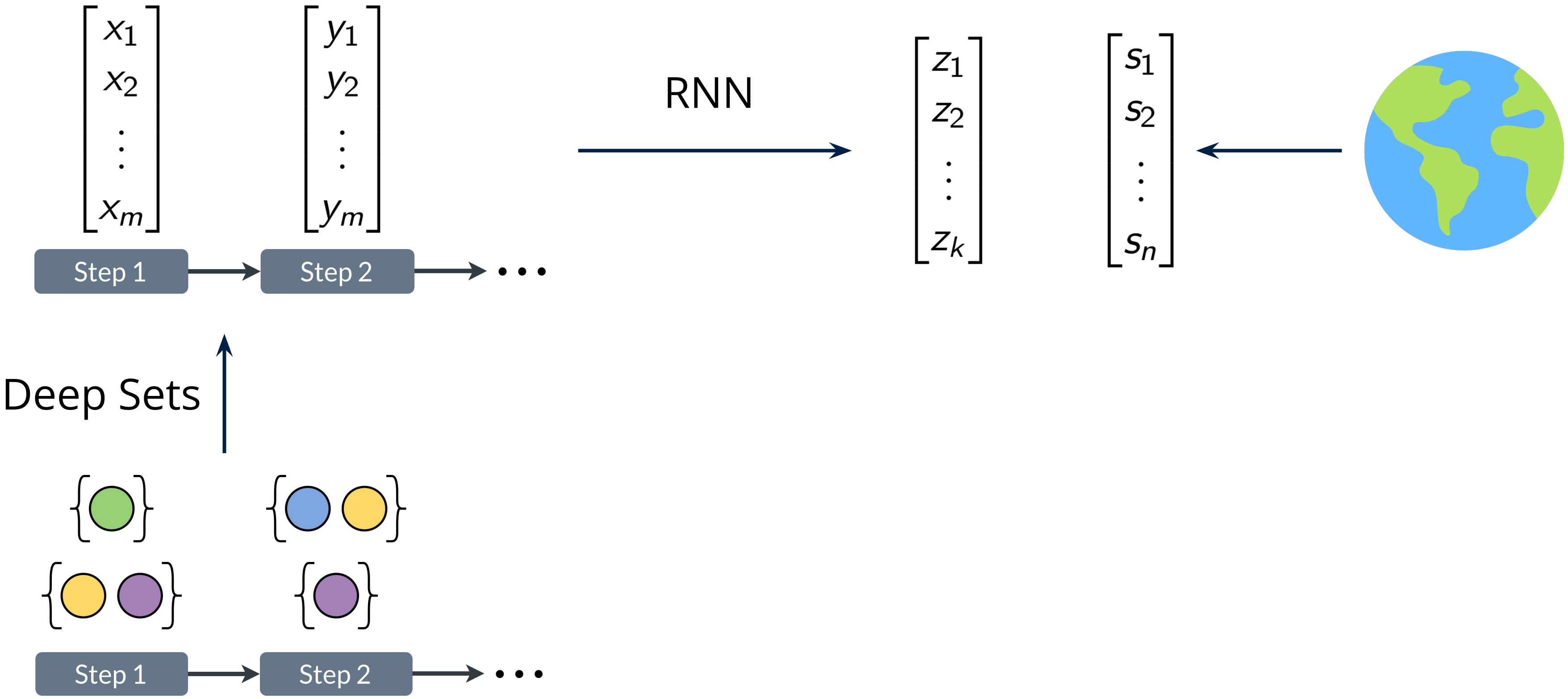
Model architecture



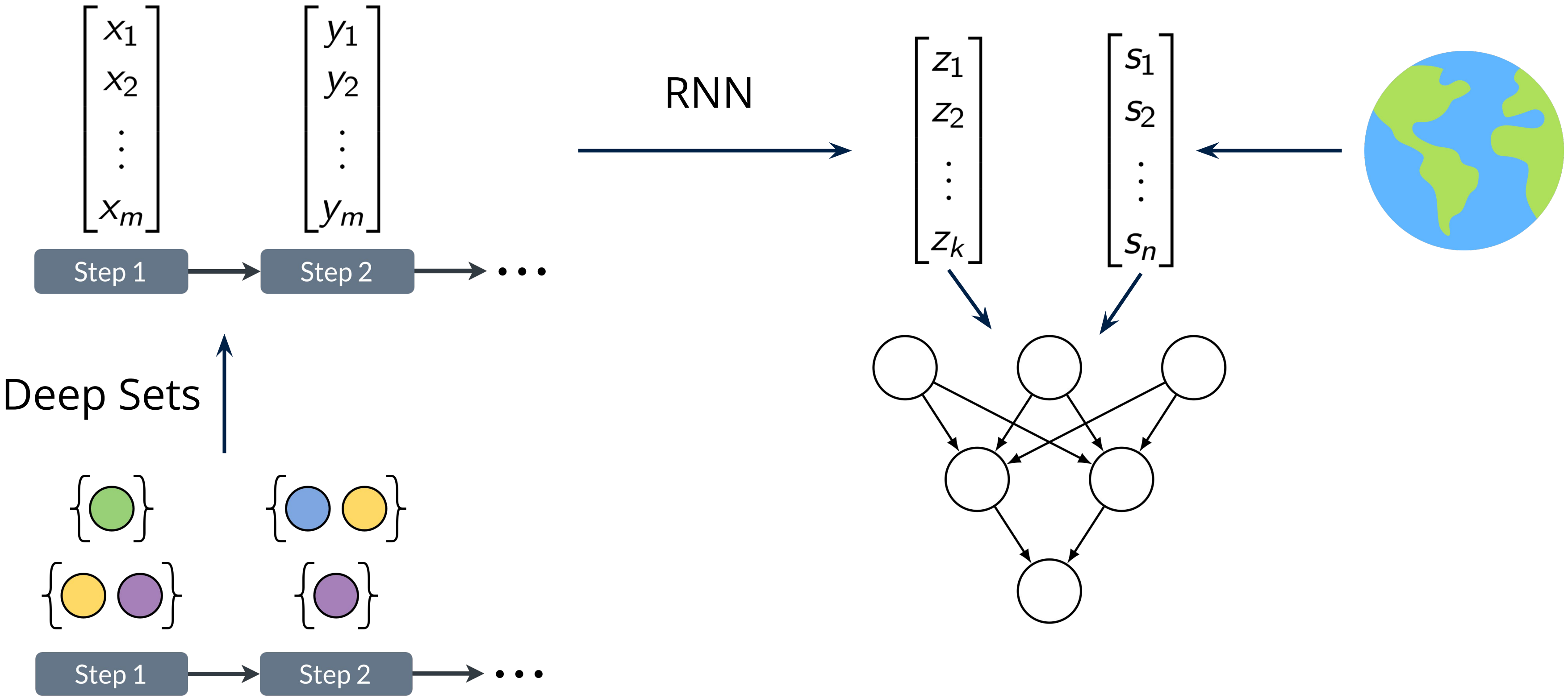
Model architecture



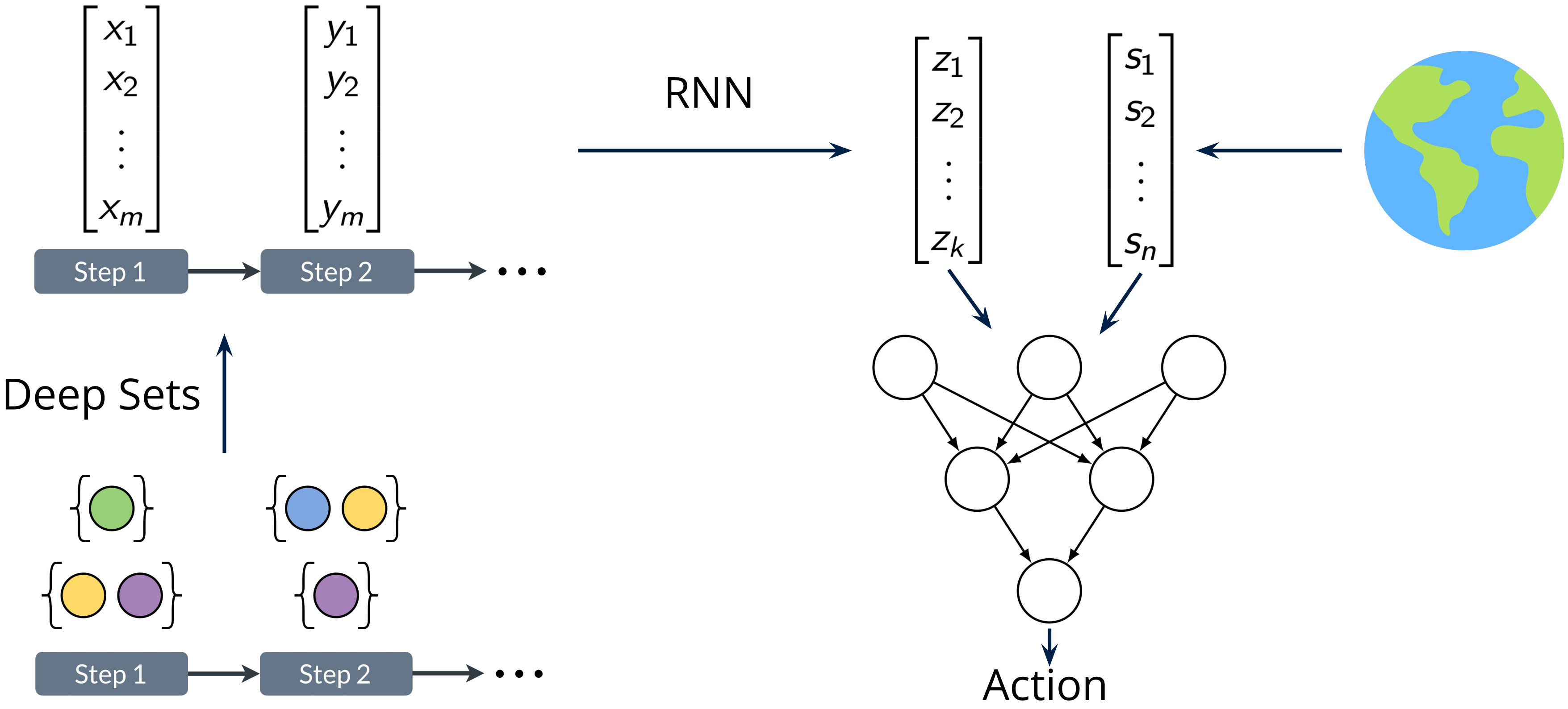
Model architecture



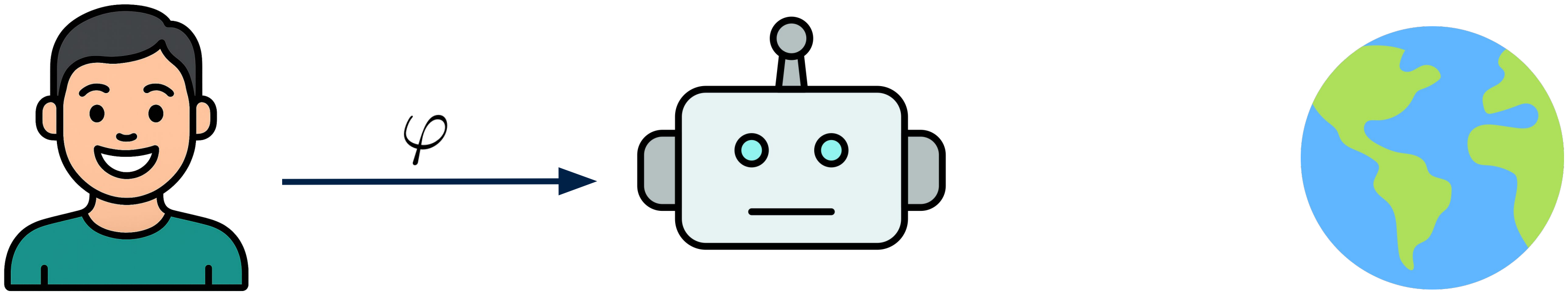
Model architecture



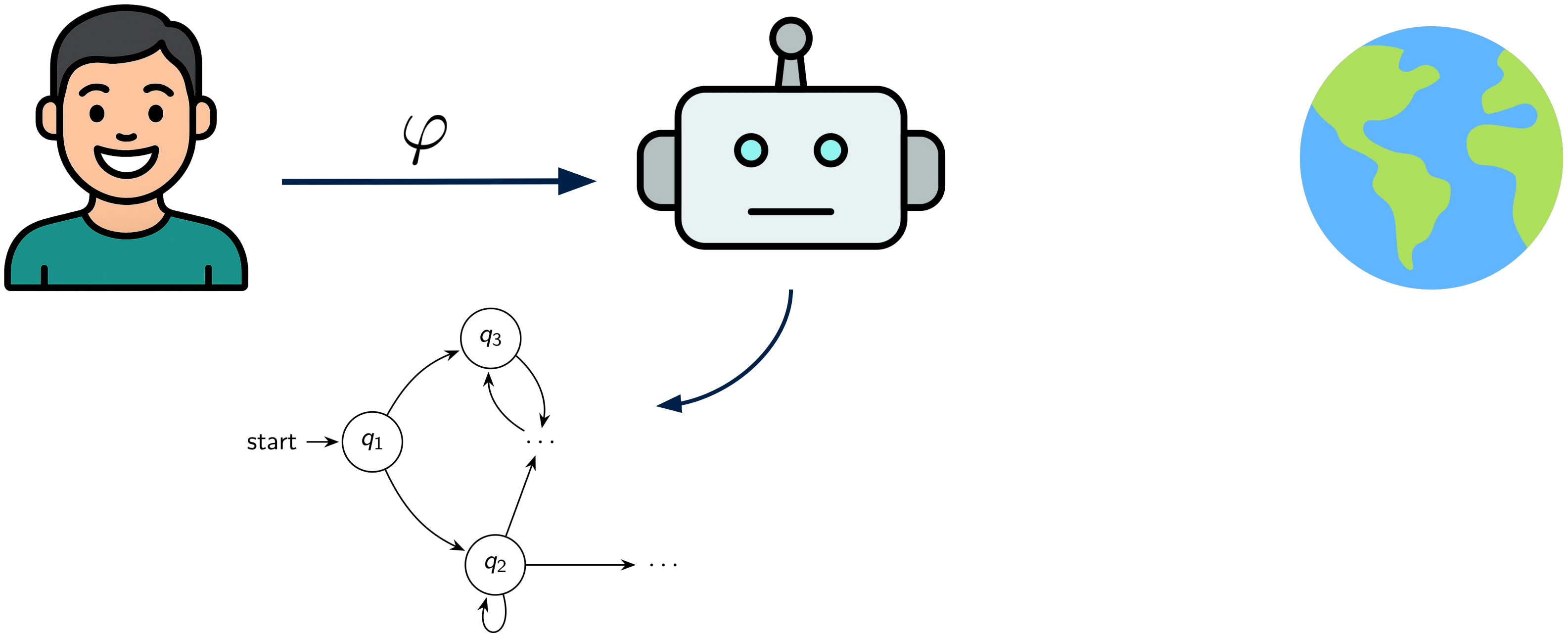
Model architecture



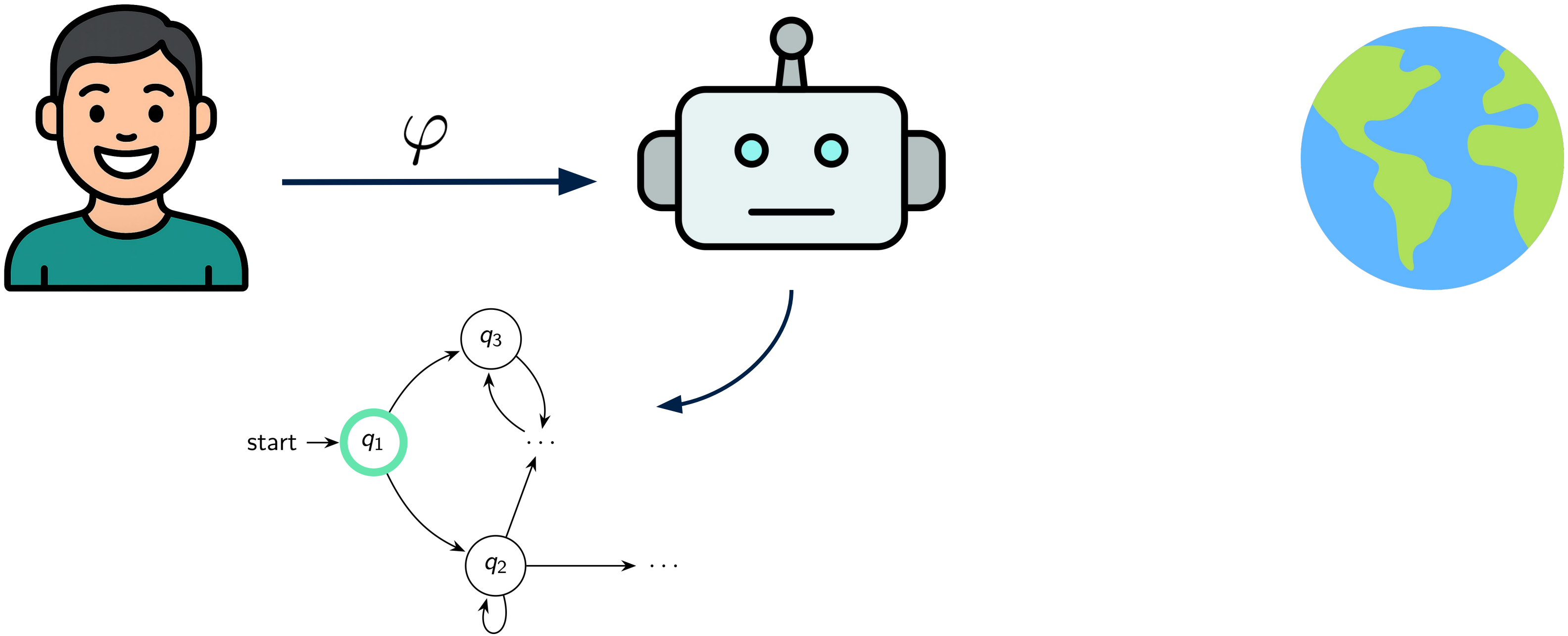
Test-time policy execution



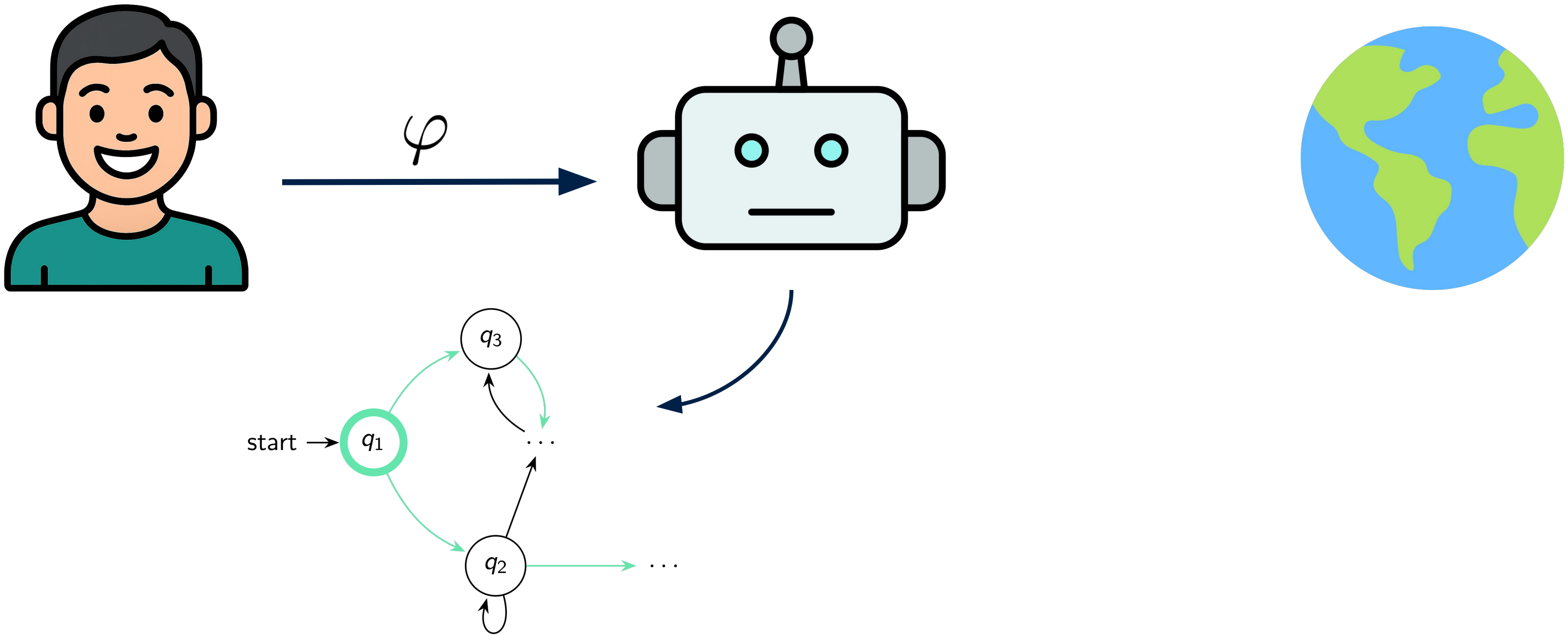
Test-time policy execution



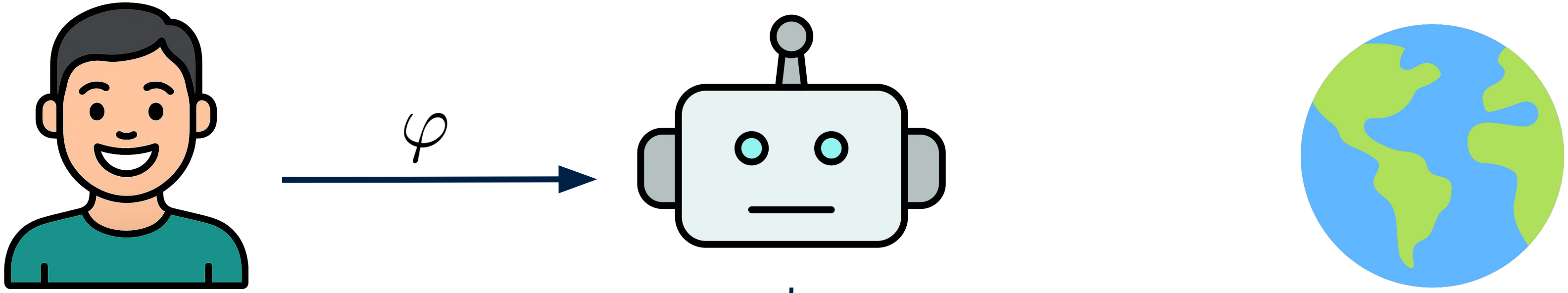
Test-time policy execution

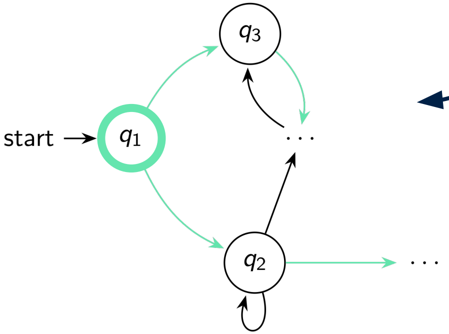


Test-time policy execution



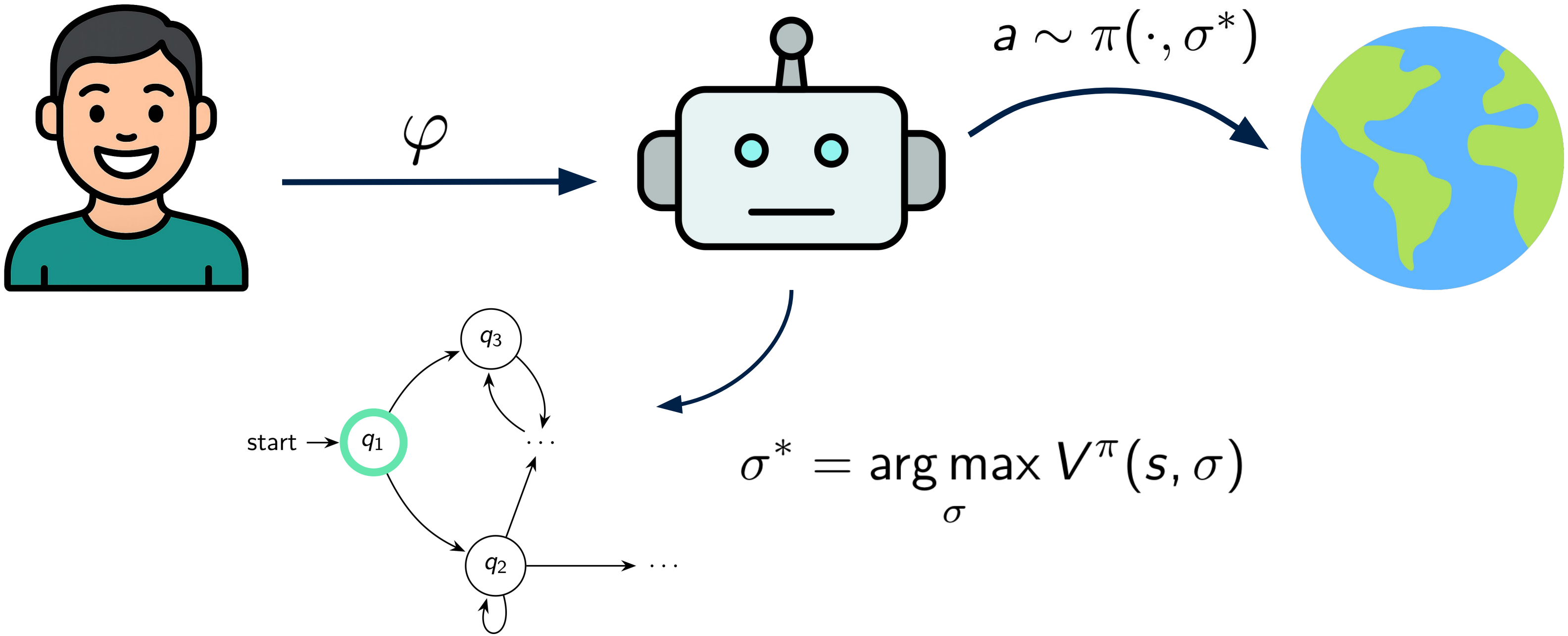
Test-time policy execution



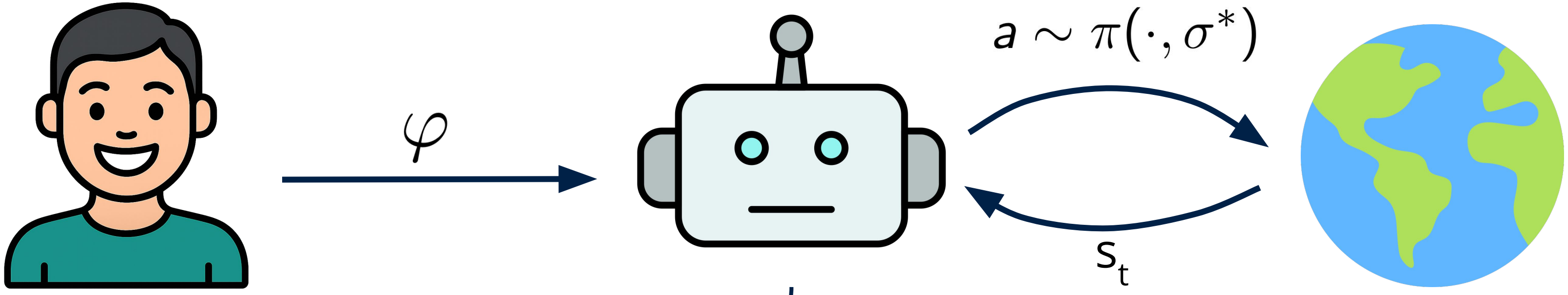


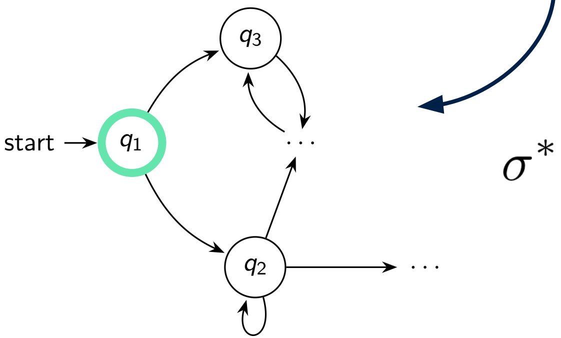
$$\sigma^* = \arg\max_{\sigma} V^{\pi}(s, \sigma)$$

Test-time policy execution



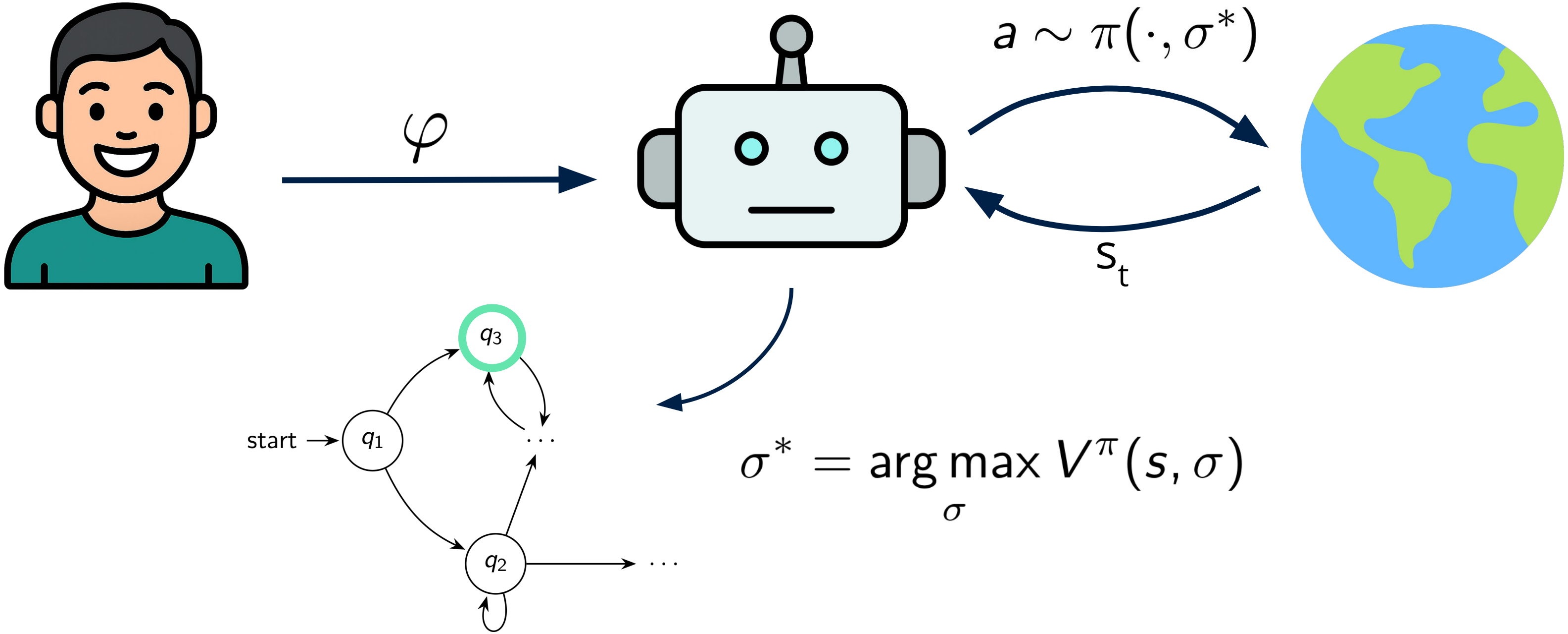
Test-time policy execution





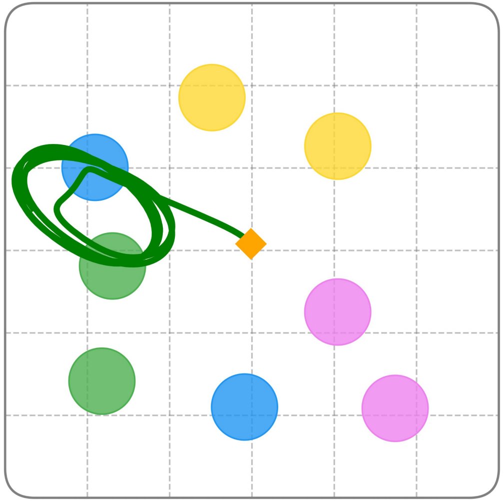
$$\sigma^* = \arg\max_{\sigma} V^{\pi}(s, \sigma)$$

Test-time policy execution

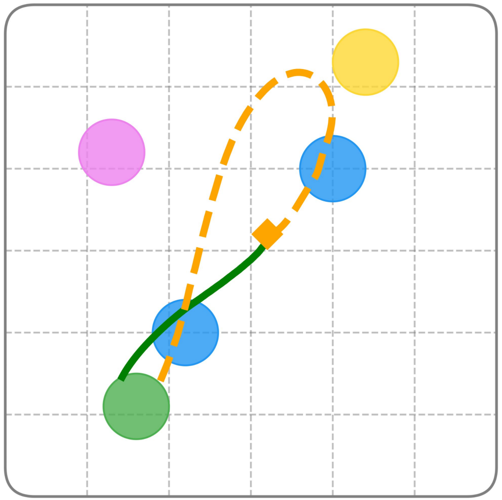


Discussion & Results

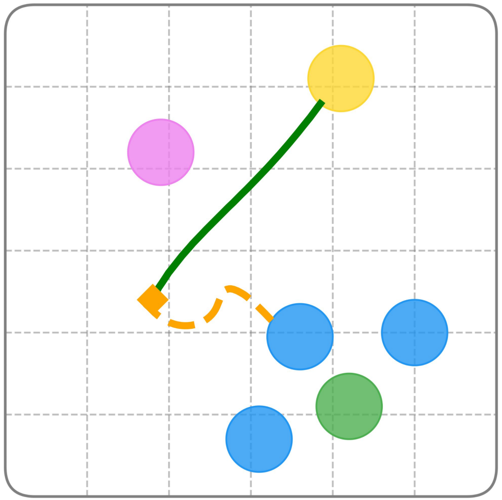
Discussion



Infinite-horizon tasks

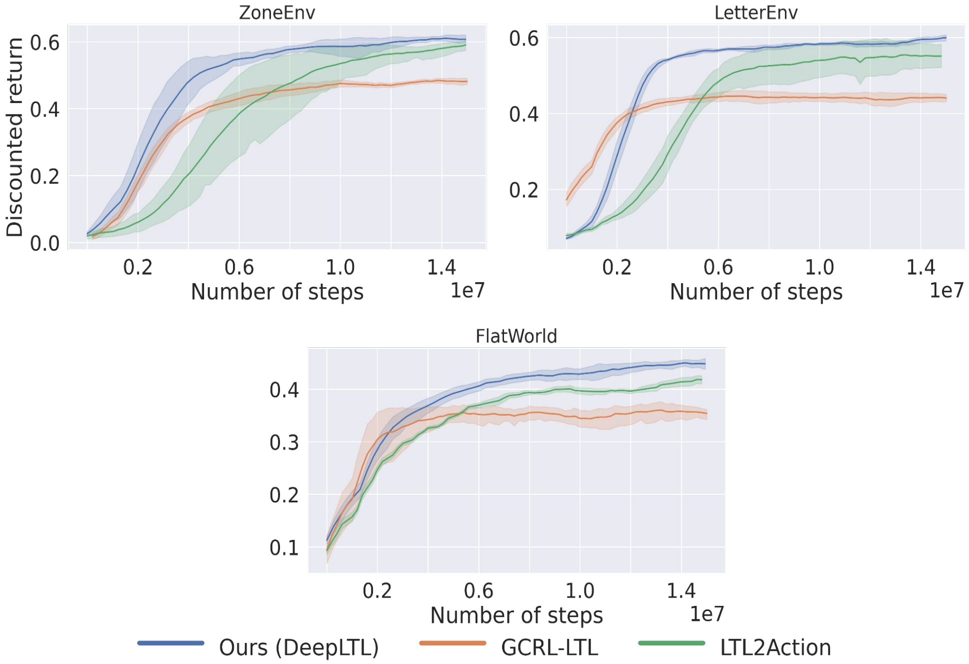


Optimality



Safety

Results



		LTL2Action	GCRL-LTL	DeepLTL
LetterWorld	φ_1	$0.75_{\pm 0.18}$	$0.94_{\pm 0.05}$	$\mathbf{1.00}_{\pm 0.00}$
	φ_2	$0.79_{\pm 0.10}$	$0.94_{\pm 0.03}$	$\mathbf{0.98}_{\pm 0.00}$
	φ_3	$0.41_{\pm 0.14}$	$\mathbf{1.00}_{\pm 0.00}$	$\mathbf{1.00}_{\pm 0.00}$
	φ_4	$0.72_{\pm 0.17}$	$0.82_{\pm 0.07}$	$\mathbf{0.97}_{\pm 0.01}$
	φ_5	$0.44_{\pm 0.26}$	$\mathbf{1.00}_{\pm 0.00}$	$\mathbf{1.00}_{\pm 0.00}$
ZoneEnv	φ_6	$0.60_{\pm 0.20}$	$0.85_{\pm 0.03}$	$\mathbf{0.92}_{\pm 0.06}$
	φ_7	$0.14_{\pm 0.18}$	$0.85_{\pm 0.05}$	$\mathbf{0.91}_{\pm 0.03}$
	φ_8	$0.67_{\pm 0.26}$	$0.89_{\pm 0.04}$	$\mathbf{0.96}_{\pm 0.04}$
	φ_9	$0.69_{\pm 0.22}$	$0.87_{\pm 0.02}$	$\mathbf{0.90}_{\pm 0.03}$
	φ_{10}	$0.66_{\pm 0.19}$	$0.85_{\pm 0.02}$	$\mathbf{0.91}_{\pm 0.02}$
	φ_{11}	$0.93_{\pm 0.07}$	$0.89_{\pm 0.01}$	$\mathbf{0.98}_{\pm 0.01}$
FlatWorld	φ_{12}	$\mathbf{1.00}_{\pm 0.00}$	$0.82_{\pm 0.41}$	$\mathbf{1.00}_{\pm 0.00}$
	φ_{13}	$0.63_{\pm 0.50}$	$0.00_{\pm 0.00}$	$\mathbf{1.00}_{\pm 0.00}$
	φ_{14}	$0.71_{\pm 0.40}$	$0.73_{\pm 0.41}$	$\mathbf{0.98}_{\pm 0.01}$
	φ_{15}	$0.07_{\pm 0.02}$	$0.73_{\pm 0.03}$	$\mathbf{0.86}_{\pm 0.01}$
	φ_{16}	$0.56_{\pm 0.35}$	$0.64_{\pm 0.08}$	$\mathbf{1.00}_{\pm 0.01}$

Further resources

Website: deep-ltl.github.io

arXiv: arxiv.org/abs/2410.04631

GitHub: mathiasj33/deep-ltl

💻 mathias-jackermeier.me

✉️ mathias.jackermeier
@cs.ox.ac.uk

mathiasj33

@m_jackermeier