Incorporating domain knowledge into reinforcement learning

Cathy Wu

1.041/1.200 Transportation: Foundations and Methods

Readings

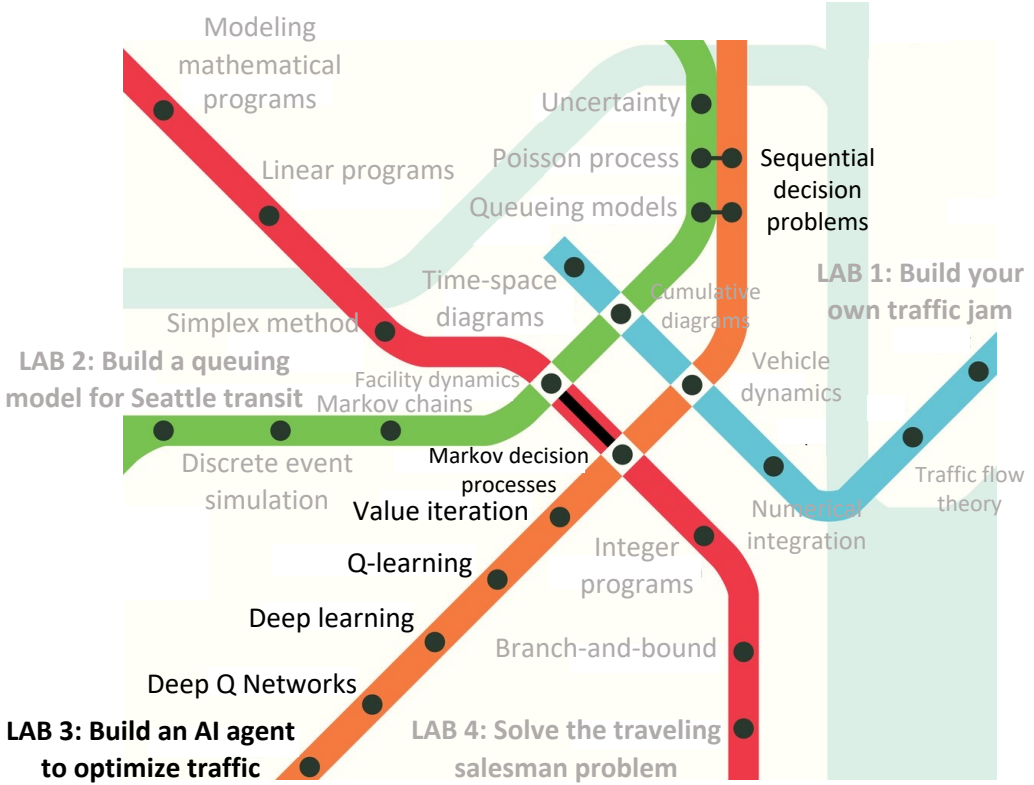
1. Miller, Tim. **Introduction to reinforcement learning**. 2024.
   - Reward shaping [URL]
2. (Optional) Henderson, Islam, Bachman, Pineau, Precup, Meger, “Deep Reinforcement Learning That Matters,” AAAI, 2018.

Unit 3: Decision Making Under Uncertainty

Unit 3

Optimizing

Multi-stage



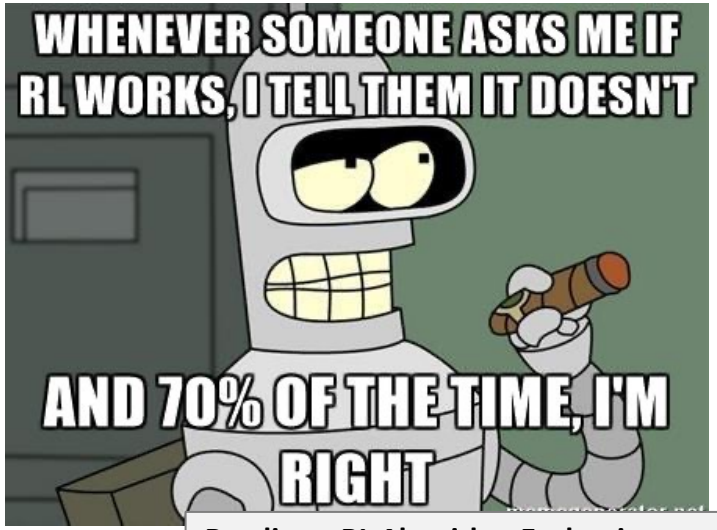
Outline

1. Why is domain knowledge important for RL?
2. Strategies to mitigate RL sensitivity
3. Reward shaping
4. Imitation learning
5. Residual policy learning
6. Learning-guided optimization

Outline

1. **Why is domain knowledge important for RL?**
   a. RL sensitivity
2. Strategies to mitigate RL sensitivity
3. Reward shaping
4. Imitation learning
5. Residual policy learning
6. Learning-guided optimization

“Does RL work?”



[1]

Solver Effectiveness (performance & speed)

Deep RL + domain knowledge, compute, manual trial-and-error

Off-the-shelf deep RL

Readings: RL Algorithm Evaluation

- Various other ways that RL is sensitive [Henderson, et al., 2018; Engstrom, et al., 2020; Andrychowicz et al., 2021; Adkins, et al., 2024]
- RL risks overfitting on benchmarks [Whiteson, et al., 2011]
- RL empirical design [Patterson, et al., 2023; Jordan, et al., 2024]

Low (Automatic)

Solver Design Cost (people & compute hours)

High (Manual)

[1] Irpan, Alex. "Deep Reinforcement Learning Doesn't Work Yet." 14 Feb. 2018, www.alexirpan.com/2018/02/14/rl-hard.html.

Does deep reinforcement learning work?

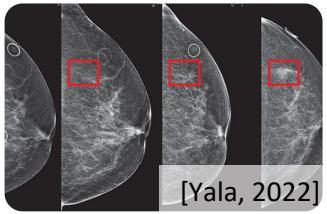
Yes! Groundbreaking results

[Bellemare, 2020]

Google Loon

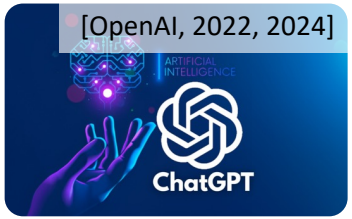
[Feng, 2023]

AV safety validation



Breast cancer screening

[OpenAI, 2022, 2024]



ChatGPT

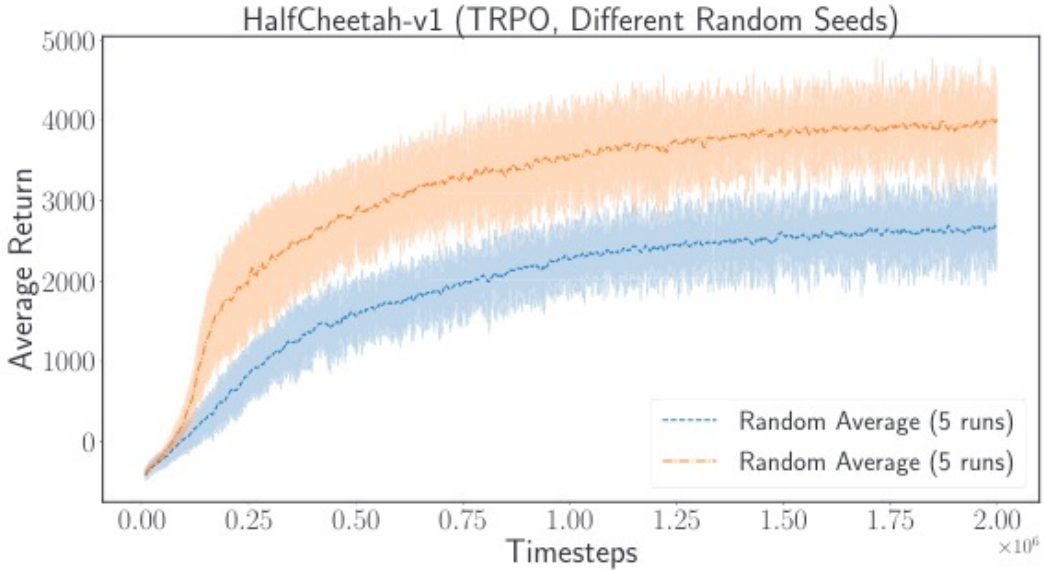
No! Not consistently

RL is highly sensitive to… [1-4]

- Network architecture
- Inherited codebase
- Code-level optimizations
- Domains / Benchmarks
- Random seed
- Method hyperparameters
- MDP specification (observation, discount rate, frame skip, etc.)
- Task variations

[1] Henderson, Islam, Bachman, Pineau, Precup, Meger, "Deep Reinforcement Learning That Matters," *AAAI*, 2018.
[2] L. Engstrom et al., "Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO," ICLR, 2020.
[3] M. Andrychowicz et al., "What matters for on-policy deep actor-critic methods? A large-scale study," ICLR, 2021.
[4] Jayawardana, Tang, Li, Suo, **Wu**. "The Impact of Task Underspecification in Evaluating Deep Reinforcement Learning," NeurIPS, 2022.

Example: Sensitivity of RL to random seeds



Difference is “statistically significant”

Henderson, Islam, Bachman, Pineau, Precup, Meger, **Deep Reinforcement Learning That Matters**, *AAAI Conference on Artificial Intelligence*, 2018.

Example: Sensitivity of RL to network architecture

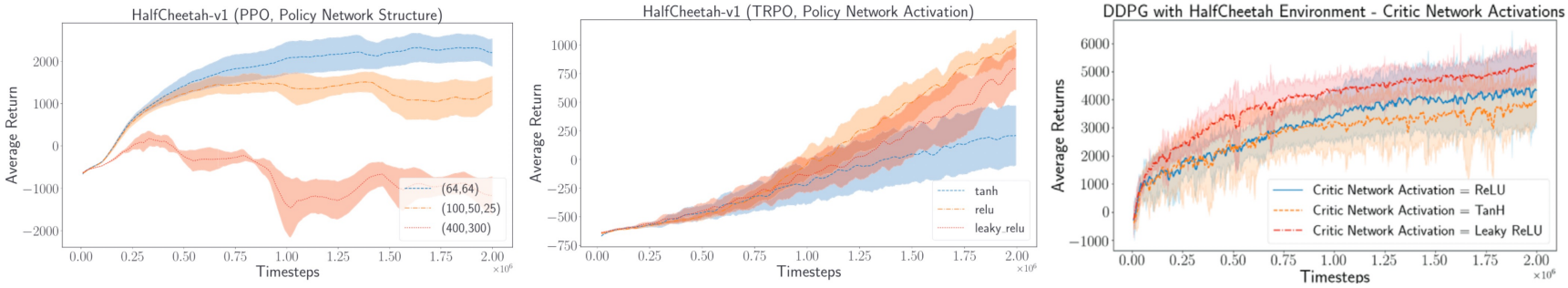


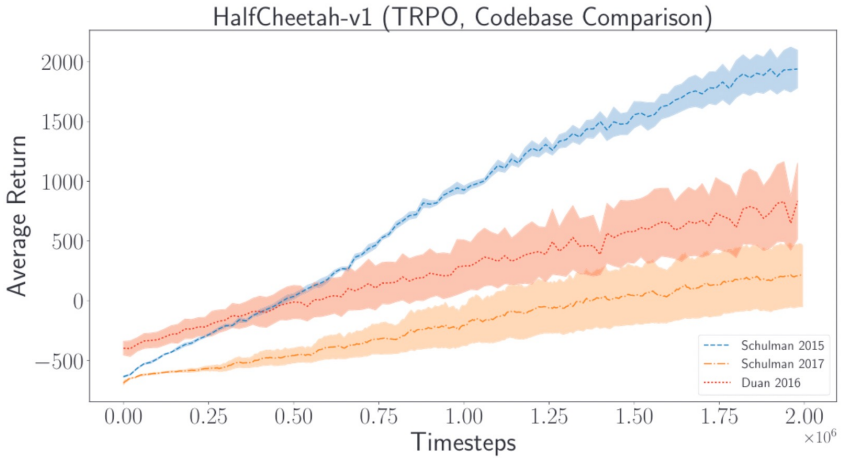
Figure 2: Significance of Policy Network Structure and Activation Functions PPO (left), TRPO (middle) and DDPG (right).

Half Cheetah

Henderson, Islam, Bachman, Pineau, Precup, Meger, **Deep Reinforcement Learning That Matters**, *AAAI Conference on Artificial Intelligence*, 2018.

Wu

Example: Sensitivity of RL to codebase



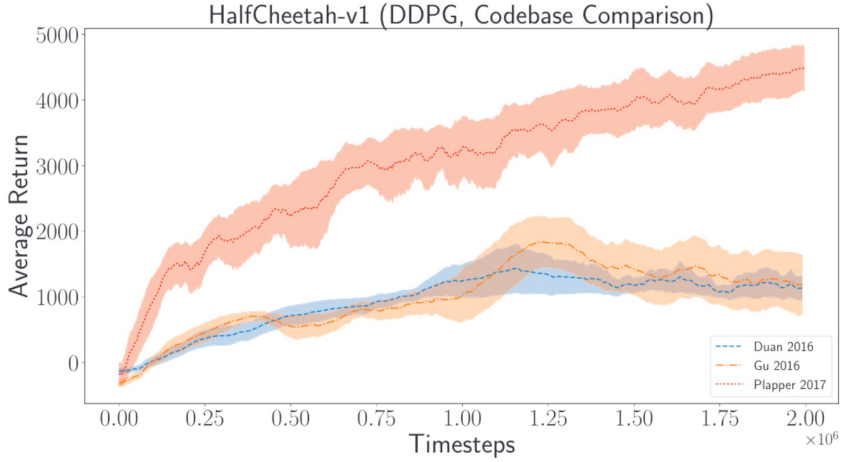


Figure 6: TRPO codebase comparison using our default set of hyperparameters (as used in other experiments).

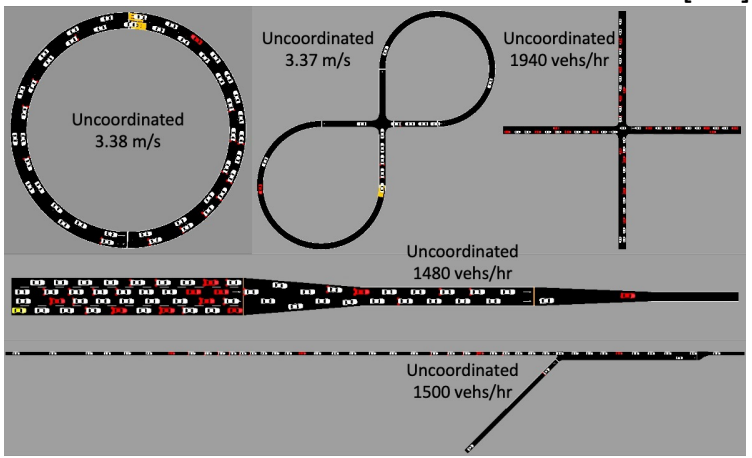
Half Cheetah

Henderson, Islam, Bachman, Pineau, Precup, Meger, **Deep Reinforcement Learning That Matters**, *AAAI Conference on Artificial Intelligence*, 2018.

Wu

Summary of my first two years at MIT

Success during PhD [1-3]



≈10 **unsuccessful** applications of RL

Success with enough engineering

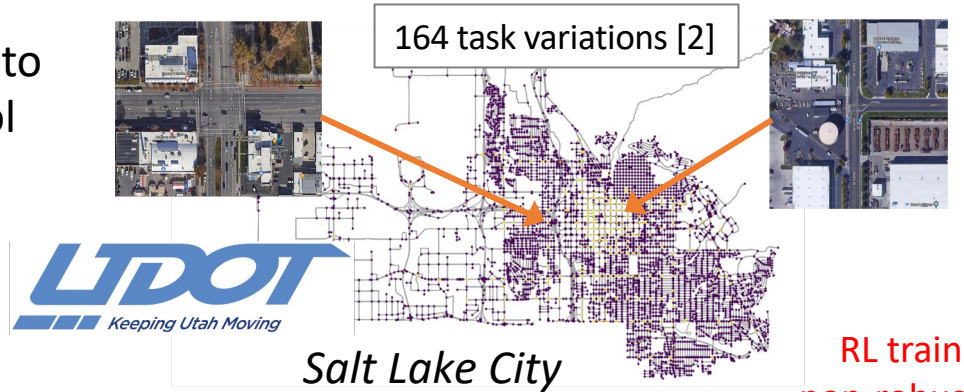
[1] **Wu**, Kreidieh, Vinitsky, Bayen, "Emergent behaviors in mixed-autonomy traffic," in *CoRL*, 2017.
[2] **Wu**, Kreidieh, Parvate, Vinitsky, Bayen, "Flow: A modular learning framework for mixed autonomy traffic," *IEEE T-RO*, 2021.
[3] Yan, Kreidieh, Vinitsky, Bayen, **Wu**, "Unified automatic control of vehicular systems with reinforcement learning," *IEEE T-ASE*, 2022.

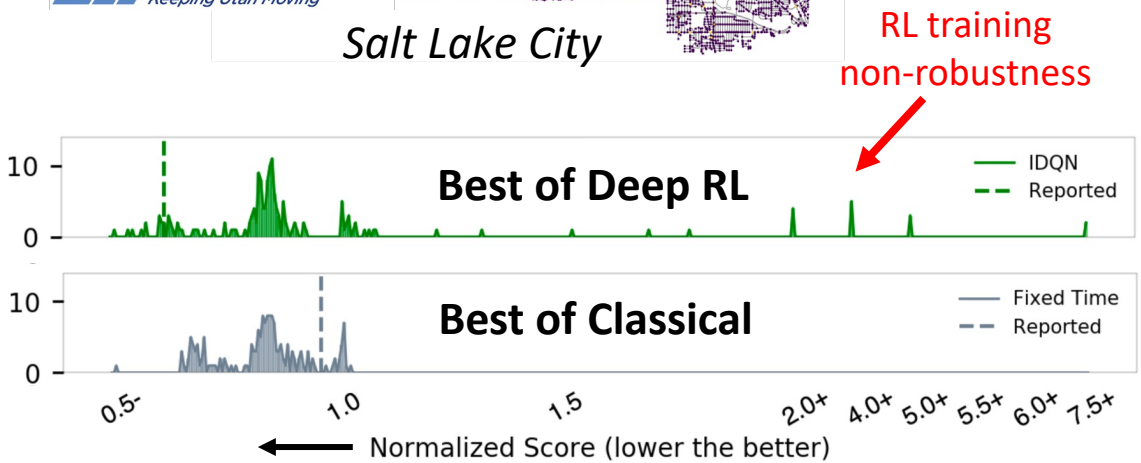
RL non-robustness to task variation

Deep RL methods are **not robust** to variations in traffic signal control

	Method	Relative Delay (↓)
Classical	**Fixed Time**	**0.82**
	Max Pressure	0.98
Deep RL	IDQN	1.06
	IPPO	1.69
	MPLight	1.11
	MPLight*	1.49

Worse than fixed time

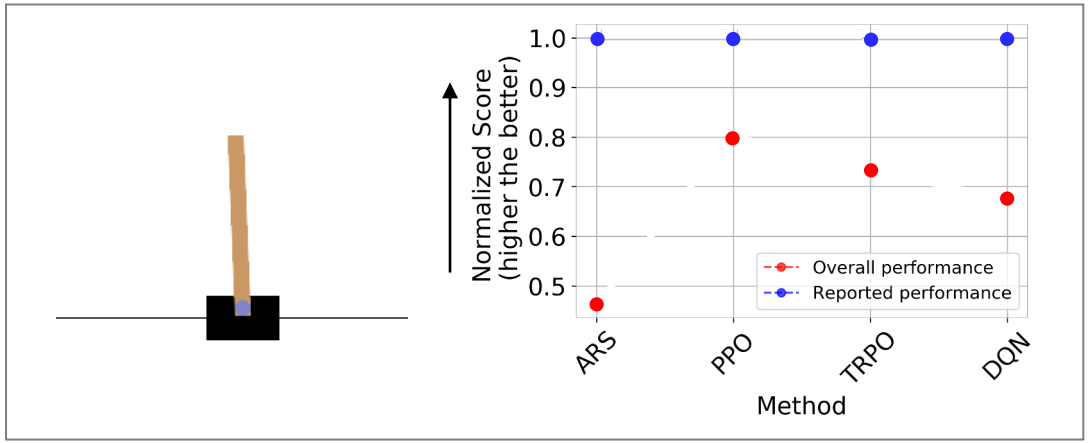




[1] Ault, Sharon. “Reinforcement learning benchmarks for traffic signal control.” Advances in Neural Information Processing Systems (NeurIPS), 2021.
[2] Qu*, Valiveru*, Tang, Jayawardana, Freydt, **Wu**. “What is a Typical Signalized Intersection in a City? A Pipeline for Intersection Data Imputation from OpenStreetMap.” TRB, 2023.
Jayawardana, Tang, Li, Suo, **Wu**. “The Impact of Task Underspecification in Evaluating Deep Reinforcement Learning.” Advances in Neural Information Processing Systems (NeurIPS), 2022.

RL non-robustness to task variation

Similar findings with general deep RL methods & standard control benchmarks



Deep RL methods are **not robust** to variations in Cartpole

Deep RL methods are highly sensitive to task variation.

Yet, task variation is everywhere in real applications.

*Reported performance is reproduced from common benchmark task specification.

Jayawardana, Tang, Li, Suo, **Wu**. "The Impact of Task Underspecification in Evaluating Deep Reinforcement Learning." Advances in Neural Information Processing Systems (NeurIPS), 2022.

Outline

1. Why is domain knowledge important for RL?
2. **Strategies to mitigate RL sensitivity**
3. Reward shaping
4. Imitation learning
5. Residual policy learning
6. Learning-guided optimization

Strategies for handling RL sensitivity

- Prefer using established libraries
  - Often less sensitive than rolling your own (when getting started)
  - Standardize, standardize, standardize
  - Benchmarks
  - RL codebases
  - Hyperparameter tuners
  - …

Strategies for handling RL sensitivity

- More data
  - Large batch sizes
  - Train multiple times (same hyperparameters)
  - Train with different hyperparameters (hyperparameter tuning)
  - Write/use fast vectorized simulators

Example computation costs (recall Rainbow)

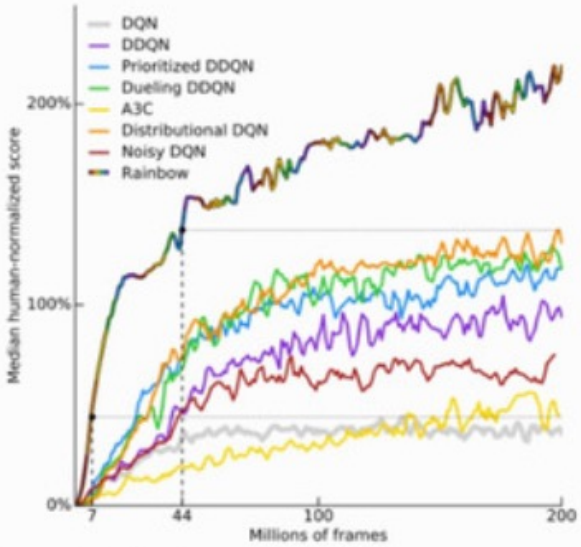


Figure 1: **Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final performance. Curves are smoothed with a moving average over 5 points.

The cost of Rainbow:

- ~5 days to train a game on a P100
- 57 games
- 5 independent seeds
- P100 costs about US$6000

Lower-bound of cost:

- 34,200 GPU hours
- 1425 days
- Total cost: EXPENSIVE

Obando-Ceron J. S., Castro P. S. Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research. ICML, 2021.

Example computation costs

- 1 TPU ≈ 5-30 GPUs
- AlphaGo computation for real-time game play
  - 50 TPUs on Google Cloud
  - Searches ~50 moves deep
  - ~100,000 positions per second
- MuZero training cost: ≈220 GPU-years
  - (3.8 GPU-years per Atari game) x (57 games)
  - “For each board game, we used 16 TPUs for training and 1,000 TPUs for self-play. For each game in Atari, in the 20 billion frame setting we used 8 TPUs for training and 32 TPUs for self-play.”

Sunnyvale, CA (Image courtesy Dylan Cutler, Google)

Strategies for handling RL sensitivity

- Don’t make learning harder than it needs to be → Environment design
- Don’t learn decision strategies that are already known → Incorporate w/ RL

Outline

1. Why is domain knowledge important for RL?
2. Strategies to mitigate RL sensitivity
3. **Reward shaping**
4. Imitation learning
5. Residual policy learning
6. Learning-guided optimization

Environment design

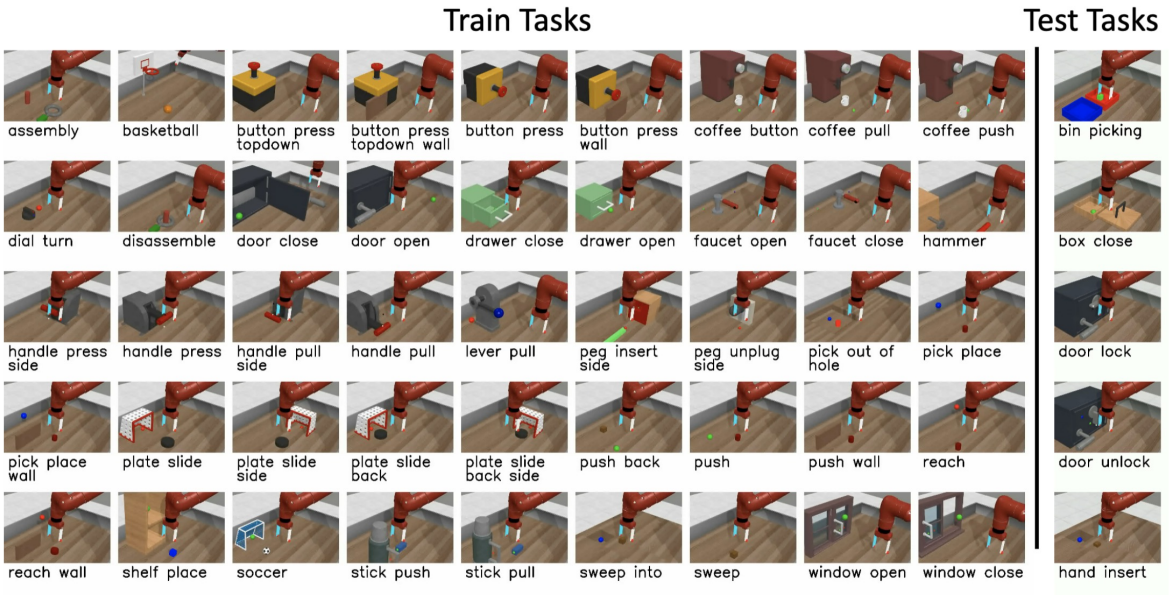
Natural places to inject domain knowledge is to consider the computational tractability of different modeling choices:

- State space
- Action space
- Horizon
- Discount
- **Reward function**

See Lecture 14

Reward functions matter

- Example from Robotics (MetaWorld Benchmark)




```python
    def compute_reward(
        self, actions: npt.NDArray[Any], obs: npt.NDArray[np.float64]
    ) -> tuple[float, float, float, float, float, float]:
        assert (
            self._target_pos is not None and self.obj_init_pos is not None
        ), "`reset_model()` must be called before `compute_reward()`."
        if self.reward_function_version == "v2":
            _TARGET_RADIUS: float = 0.05
            tcp = self.tcp_center
            obj = obs[4:7]
            tcp_opened: float = obs[3]
            target = self._target_pos

            obj_to_target = float(np.linalg.norm(obj - target))
            in_place_margin = float(np.linalg.norm(self.obj_init_pos - target))
            in_place = reward_utils.tolerance(
                obj_to_target,
                bounds=(0, _TARGET_RADIUS),
                margin=in_place_margin - _TARGET_RADIUS,
                sigmoid="long_tail",
            )

            tcp_to_obj = float(np.linalg.norm(tcp - obj))
            obj_grasped_margin = float(
                np.linalg.norm(self.init_tcp - self.obj_init_pos)
            )
            object_grasped = reward_utils.tolerance(
                tcp_to_obj,
                bounds=(0, _TARGET_RADIUS),
                margin=obj_grasped_margin - _TARGET_RADIUS,
                sigmoid="long_tail",
            )

            reward = 1.5 * object_grasped

            if tcp[2] <= 0.03 and tcp_to_obj < 0.07:
                reward = 2 + (7 * in_place)

            if obj_to_target < _TARGET_RADIUS:
                reward = 10.0
            return (
                reward,
                tcp_to_obj,
                tcp_opened,
                obj_to_target,
                object_grasped,
                in_place,
            )
```

Code: https://github.com/Farama-Foundation/Metaworld/blob/master/metaworld/envs/sawyer_plate_slide_back_side_v3.py

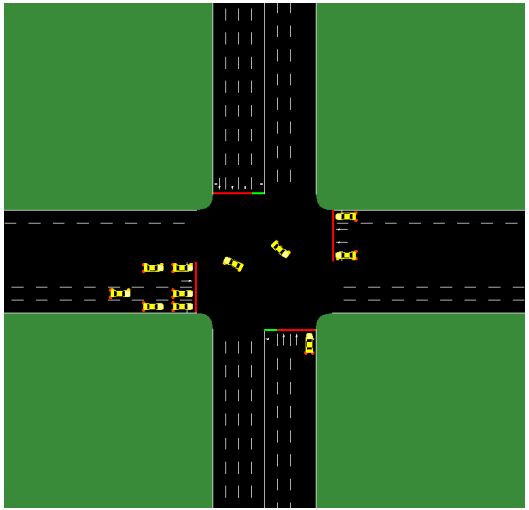
Challenge: Naïve reward functions are often sparse

- Rewards are sparse when few state/action pairs have non-zero rewards.

Adapted from Tim Miller – Introduction to RL, Reward Shaping

Recall: Choosing the right reward function

- How many steps an action influences rewards is also a function of the reward definition



Total wait time among all vehicles (over 90 minutes)

Instantaneous throughput

Instantaneous queue length

sparse

dense

effective horizon

Specifying appropriate rewards is nontrivial

- **Unintended consequences** include pushing bad rewards away in time/space or exploiting good reward logic:
  - Example: Reward average traffic speed → Learn to block on-ramp
  - Example: Exploit +1 reward for lane centering by repeatedly uncentering and re-centering
- **Reward hacking** is the phenomenon where optimizing an imperfect proxy reward function leads to poor performance according to the true reward function.

J. Skalse, N. H. R. Howe, D. Krasheninnikov, and D. Krueger, "Defining and Characterizing Reward Hacking." arXiv, Sep. 26, 2022. doi: 10.48550/arXiv.2209.13085.

Q Learning Algorithm

1. Let Q_0 be any Q-function, s be an initial state,
2. At each iteration $k = 0, 1, 2, \ldots, K$
   - Use Q_k to select an action a (use ϵ-greedy)
   - Observe next state s' and reward r
   - Update Q function: $Q_{k+1}(s,a) = (1-\eta_k)Q_k(s,a) + \eta_k\left(r + \gamma \max_{a'} Q_k(s',a')\right)$
   - $s \leftarrow s'$
3. Return the greedy policy

learning rates $\eta_k \in [0,1]$.

$$= Q_k(s,a) + \eta_k\left(r + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\right)$$

TD / bootstrap target

Current guess of value

Temporal difference (TD) error δ_k

$$\pi_K(s) = \arg\max_{a \in A} Q_K(s,a)$$

Reward shaping

- Q-learning update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

- The approach to reward shaping is not to modify the reward function or the received reward r, but to just give some additional reward for some actions:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + F(s,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Additional reward

- Shaped reward: $r + F(s,s')$
- Reward tuning: even more generally, **tuned reward**: $r + G(s,a,s')$

Additional reward

Adapted from Tim Miller – Introduction to RL, Reward Shaping

Potential-based Reward Shaping

Potential-based reward shaping is a particular type of reward shaping with nice theoretical guarantees. In potential-based reward shaping, F is of the form:

$$F(s, s') = \gamma\Phi(s') - \Phi(s)$$

We call Φ the **potential function** and $\Phi(s)$ is the **potential** of state s.

So, instead of defining $F : S \times S \to \mathbb{R}$, we define $\Phi : S \to \mathbb{R}$, which is some heuristic measure of the value of each state $s \in S$.

Theoretical guarantee: this will still converge to the optimal policy under the assumption that all state-action pairs are sampled infinitely often.

Adapted from Tim Miller – Introduction to RL, Reward Shaping

Potential-based Reward Shaping

This is quite straightforward to show as follows. Consider an episode with shaped reward G^{Φ} :

$$
\begin{aligned}
G^{\Phi} &= \sum_{i=0}^{\infty} \gamma^i (r_i + F(s_i, s_{i+1})) \\
&= \sum_{i=0}^{\infty} \gamma^i (r_i + \gamma\Phi(s_{i+1}) - \Phi(s_i)) \\
&= \sum_{i=0}^{\infty} \gamma^i r_i + \sum_{i=0}^{\infty} \gamma^{i+1}\Phi(s_{i+1}) - \sum_{i=0}^{\infty} \gamma^i \Phi(s_i) \\
&= G + \sum_{i=0}^{\infty} \gamma^i \Phi(s_i) - \Phi(s_0) - \sum_{i=0}^{\infty} \gamma^i \Phi(s_i) \\
&= G - \Phi(s_0)
\end{aligned}
$$

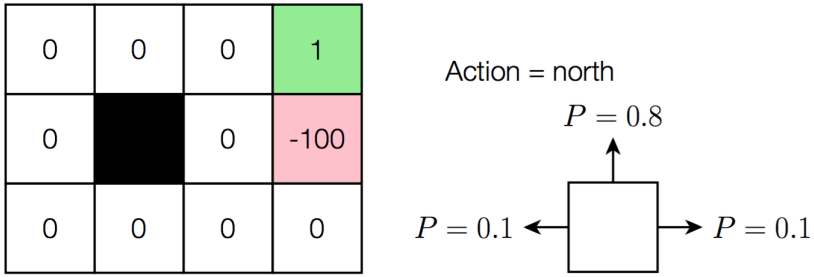
where G refers to the shaped reward for the episode, and s_0 is the starting state of the episode. What this says is that the shaped reward G^{Φ} is just the unshaped reward G minus the potential of the initial state s_0. However, because F does not depend on the actions and G^{Φ} does not depend on shaped rewards beyond the initial state, the **shaped Q function**, which we refer to as Q^{Φ}, can be defined as just $Q^{\Phi}(s, a) = Q(s, a) + \Phi(s)$. Given this, any optimal policy extracted from Q^{Φ} will be equivalent to any optimal policy extracted from Q.

However! While it provides guarantees about the end result, potential-based reward shaping may either increase or decrease the time taken to learn. A well-designed potential function decrease the time to convergence.

Adapted from Tim Miller – Introduction to RL, Reward Shaping

Example: Winter parking (with ice and potholes)

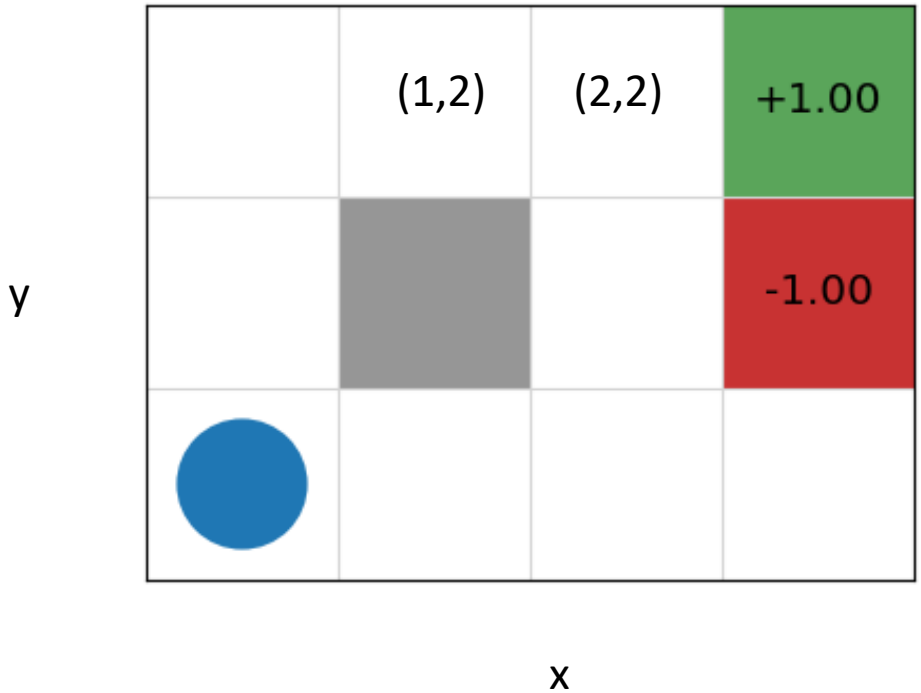
- Simple grid world with a *goal state* (green, desired parking spot) with reward (+1), a *"bad state"* (red, pothole) with reward (-100), and all other states neural (+0).
- *Omnidirectional vehicle (agent)* can head in any direction. Actions move in the desired direction with probably 0.8, in one of the perpendicular directions with.
- Taking an action that would bump into a wall leaves agent where it is.



[Source: adapted from Kolter, 2016]

Example -- Potential Reward Shaping for GridWorld

state = (x,y)



Adapted from Tim Miller – Introduction to RL, Reward Shaping

Example -- Potential Reward Shaping for GridWorld

For Grid World, we use the Manhattan distance to define the potential function, normalised by the size of the grid:

$$\Phi(s) = 1 - \frac{|x(g) - x(s)| + |y(g) - y(s)|}{width + height - 2}$$

in which $x(s)$ and $y(s)$ return the x and y coordinates of the agent respectively, g is the goal state. and $width$ and $height$ are the width and height of the grid respectively. Note that the coordinates are indexed from 0, so we subtract 2 from the denominator.

Even on the very first iteration, a greedy policy such as ϵ-greedy, will feedback those states closer to the +1 reward. From state (1,2) with $\gamma = 0.9$ if we go Right, we get:

$$\begin{aligned} F((1,2),(2,2)) &= \gamma\Phi(2,2) - \Phi(1,2) \\ &= 0.9 \cdot (1 - \tfrac{1}{5}) - (1 - \tfrac{2}{5}) \\ &= 0.12 \end{aligned}$$

Example -- Potential Reward Shaping for GridWorld

We can compare the Q-values for these states for the four different possible moves that could have been taken from (1,2), using and $\alpha = 0.1$ and $\gamma = 0.9$:

Action	r	$F(s, s')$	$\gamma \max_{a'} Q(s', a')$	New $Q(s, a)$
Up	0	$0.9(1 - \frac{2}{5}) - (1 - \frac{2}{5}) = -0.06$	0	−0.006
Down	0	$0.9(1 - \frac{2}{5}) - (1 - \frac{2}{5}) = -0.06$	0	−0.006
Right	0	$0.9(1 - \frac{1}{5}) - (1 - \frac{2}{5}) = 0.12$	0	0.012
Left	0	$0.9(1 - \frac{3}{5}) - (1 - \frac{2}{5}) = -0.24$	0	−0.024

Thus, we can see that our potential reward function rewards actions that go towards the goal and penalises actions that go away from the goal. Recall that state (1,2) is in the top row, so action Up just leaves us in state (1,2) and Down similarly because we cannot go through the walls.

But! It will not always work. Compare states (0,0) and (0,1). Our potential function will reward (0,1) because it is closer to the goal, but we know from from our value iteration example that (0,0) is a higher value state than (0,1). This is because our reward function does not consider the negative reward.

In practice, it is non-trivial to derive a perfect reward function -- it is the same problem as deriving the perfect search heuristic. If we could do this, we would not need to even use reinforcement learning -- we could just do a greedy search over the reward function.

Adapted from Tim Miller – Introduction to RL, Reward Shaping

Potential-based Reward Shaping for actions

- Potential-based reward shaping extends naturally to action information too:

$$F(s, s', a, a') = \gamma\Phi(s', a') - \Phi(s, a)$$

For a potential function $\Phi(\mathrm{s}, \mathrm{a})$.

- Example from traffic signal control:
  - state-only potential: “shorter queues are better”
  - state-action potential: “when queues look like this, serving the east-west phase is especially promising”

E. Wiewiora, G. W. Cottrell, and C. Elkan, “Principled methods for advising reinforcement learning agents,” in *Proceedings of the 20th international conference on machine learning (ICML)*, 2003.

Reward shaping demo [URL]

Outline

1. Why is domain knowledge important for RL?
2. Strategies to mitigate RL sensitivity
3. Reward shaping
4. **Imitation learning**
5. Residual policy learning
6. Learning-guided optimization

Forms of domain knowledge

- Expert demonstration data $\tau_0 = (s_0, a_0, s_1, a_1, \dots)$
  - Example: Decision logs from a train dispatcher
  - Incorporate using: Imitation learning + RL fine-tuning

- Decision rule $\pi_0 : S \to A$ or $\pi_0 : S \times A \to [0,1]$
  - Example: Intelligent Driver Model
  - Incorporate using: Residual policy learning

- Solver $f_0 : \mathcal{M} \to (a_0, a_1, \dots)$
  - Example: Search algorithm for coordinating multiple vehicles in a warehouse
  - Incorporate using: Learning-guided optimization

Imitation learning + RL fine-tuning

1. Collect expert demonstration data

$$\mathcal{D} = \left\{\tau^{(i)}\right\}_{i=1}^{N}, \qquad \tau^{(i)} = (s_0, a_0, s_1, a_1, \dots)$$

2. First train π_θ by behavior cloning

$$\min_\theta \sum_{(s,a)\sim\mathcal{D}} \ell(\pi_\theta(a|s), a)$$

for loss function ℓ: squared error (continuous actions), cross-entropy (discrete actions)

- This gives a good initial policy but may suffer from compounding error

3. Then fine-tune with RL to improve long-horizon return
   - Initialize RL training with π_θ
   - Faster and more stable than training from scratch (random initialization)

Loss functions

- Squared error (continuous actions)

$$\ell(\pi_\theta(a|s), a) = \frac{1}{2}(\pi_\theta(a|s) - a)^2$$

where $\pi_\theta(a|s)$ is the predicted action and a is the expert action

- Cross-entropy (discrete actions)

$$\ell(\pi_\theta(a|s), a) = -\log \pi_\theta(a|s)$$

where $\pi_\theta(a|s)$ is the probability the policy assigns to expert action a

Outline

1. Why is domain knowledge important for RL?
2. Strategies to mitigate RL sensitivity
3. Reward shaping
4. Imitation learning
5. **Residual policy learning**
6. Learning-guided optimization

Residual policy learning

- **Example**: Suppose your goal is to solve a cooperative driving control problem

Objective: Reduce emissions and smooth traffic → $\pi^*(s, c)$ → Longitudinal acceleration →

- **Observation**: Many decision rules exist or are easy to define (e.g., IDM), albeit simplistic.

- **Method**: Residual policy learning can be leveraged to make use of those policies with a learned policy for error correction [1,2].

$$\pi^*(s) = \pi_n(s) + f_\theta(s)$$

action to execute — nominal action — residual action

[1] Silver et al. Residual policy learning, 2018
[2] Johannink et al. Residual reinforcement learning for robot control. International Conference on Robotics and Automation (ICRA), 2019

Residual policy learning for task variations

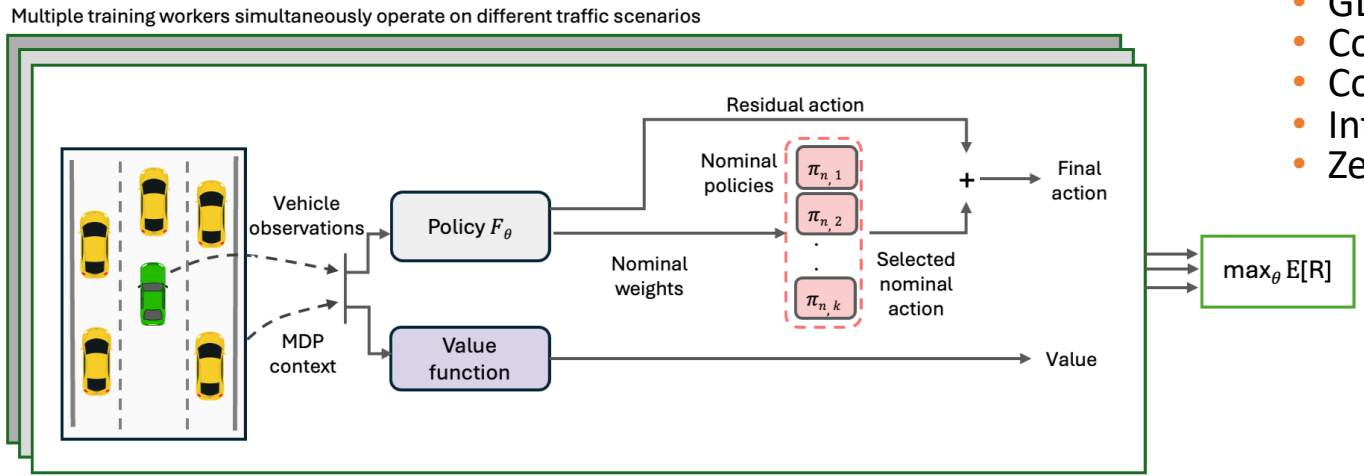
Extend residual policy learning with,

1. Context conditioning to explicitly account for task variations.

$$\pi^*(s,c) = \pi_n(s) + f_\theta(s,c)$$

2. Combine multiple nominal policies in a mixture of experts architecture to account for nominal policies that are specialized in subspaces of the task variation space.

$$\pi^*(s,c) = \sum_{k=1}^{K} g_k(s,c)\pi_n^k(s) + f_\theta(s,c)$$



- Nominal policies:
  - GLOSA
  - Constant acceleration
  - Constant deceleration
  - Intelligent Driver Model
  - Zero action

V. Jayawardana, S. Li, **C. Wu**, Y. Farid, and K. Oguchi, "Generalizing Cooperative Eco-driving via Multi-residual Task Learning," in *ICRA,* 2024. doi: 10.1109/ICRA57147.2024.10610586.
V. Jayawardana, S. Li, Y. Farid, and **C. Wu**, "Multi-residual Mixture of Experts Learning for Cooperative Control in Multi-vehicle Systems," arXiv. doi: 10.48550/arXiv.2507.09836.

What's the policy significance of Intelligent Transportation Systems (ITS)?

Source: Audi

- Case study: **Eco-driving at signalized intersections**
- Extensively studied
  - ≈1,200 articles* (2007-2025)
  - Focus: technology ('how')
- Gap: unclear impact
  - 2-56% energy/emissions [1] and scenarios not representative

From 'how to eco-drive' to 'should we eco-drive'?

* Estimated using Google Scholar

[1] Mintsis et al. *Dynamic eco-driving near signalized intersections: Systematic review and future research directions*. Journal of Transportation Engineering, Part A, 2020

Key Challenge: Scale

Complexity of scenarios
(factors that influence emissions)

X

Number of scenario cases
(1+ million scenarios)

Vehicle control

Speed choice, Accelerating / decelerating, Idling vs coasting, Lane changing, Route choice

Fleet composition

Eco-driving adoption rate, Powertrain & fuel type, Vehicle model & age, Vehicle load, Air conditioning, Vehicle maintenance

Physical environment

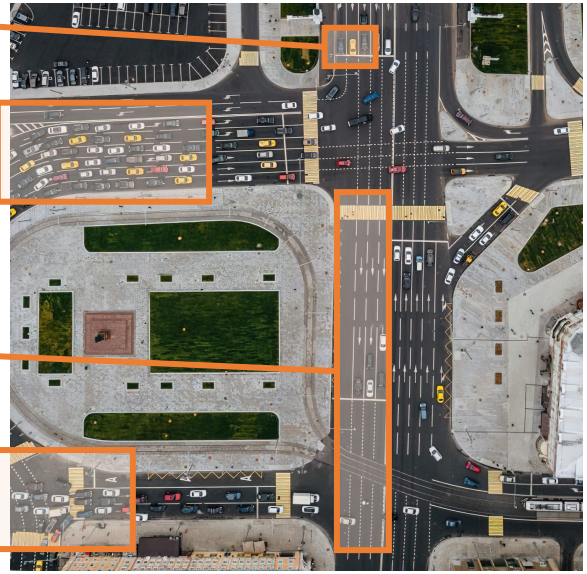
Humidity, Temperature, Weather conditions, Seasonal effects, Aerodynamic drag

Infrastructure & geometry

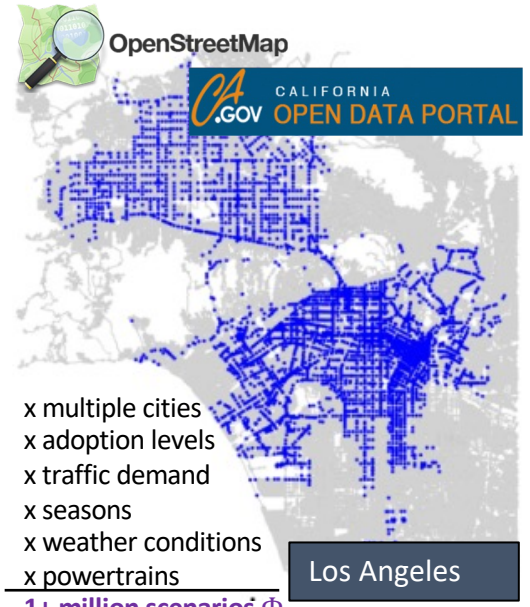
Road grade, Number of lanes, Turn-lane configurations, Lane length, Road surface & quality

Traffic dynamics

Traffic demand & OD patterns, Traffic signal timing, Speed limit, Pedestrians and cyclists, Nearby intersections



4676 Signalized Intersections [1]



x multiple cities
x adoption levels
x traffic demand
x seasons
x weather conditions
x powertrains

1+ million scenarios Φ

Issue: No available off-the-shelf solvers

Approach: Residual policy learning + multi-task RL

V. Jayawardana *et al.*, "Mitigating Metropolitan Carbon Emissions with Dynamic Eco-driving at Scale," *Transportation Research Part C: Emerging Technologies*, Apr. 2025.

Findings and Policy Implications

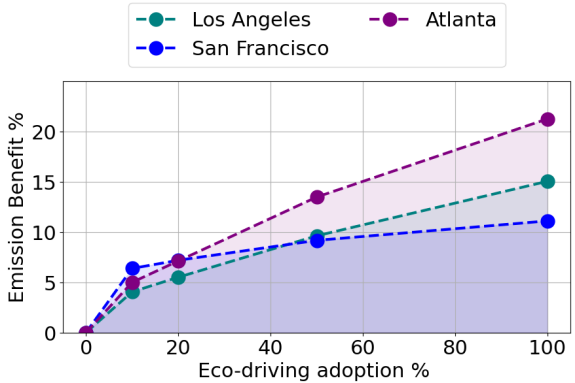
- Eco-driving is a **high-impact**, relatively **low cost** policy lever that **complements** existing interventions.

Emissions impact:
11-22% of intersection CO_2 emissions

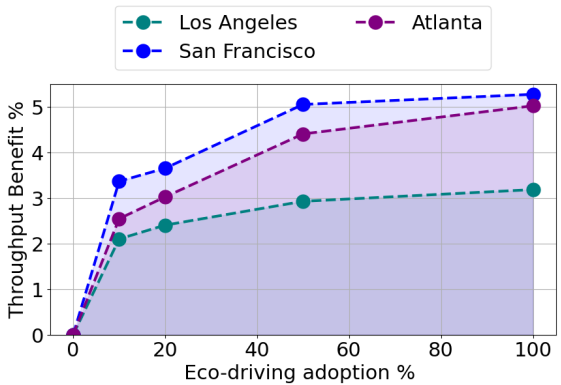
Near-term potential:
Majority of impact results from a minority of drivers and intersections

Long-term relevance:
Impact is robust through 2050 due to fleet impacts & travel demand

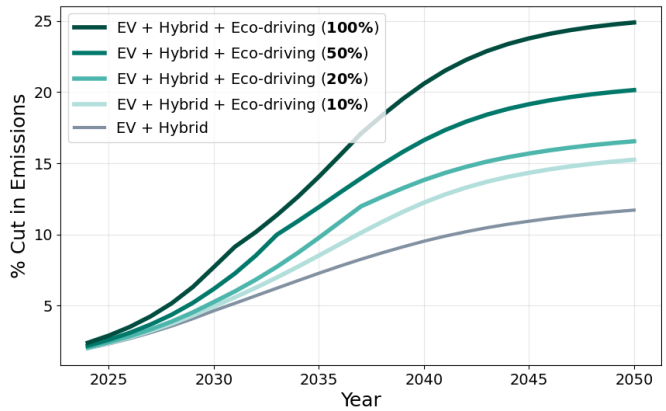
Emission reduction



Throughput increase



Projection through 2050 (LA)

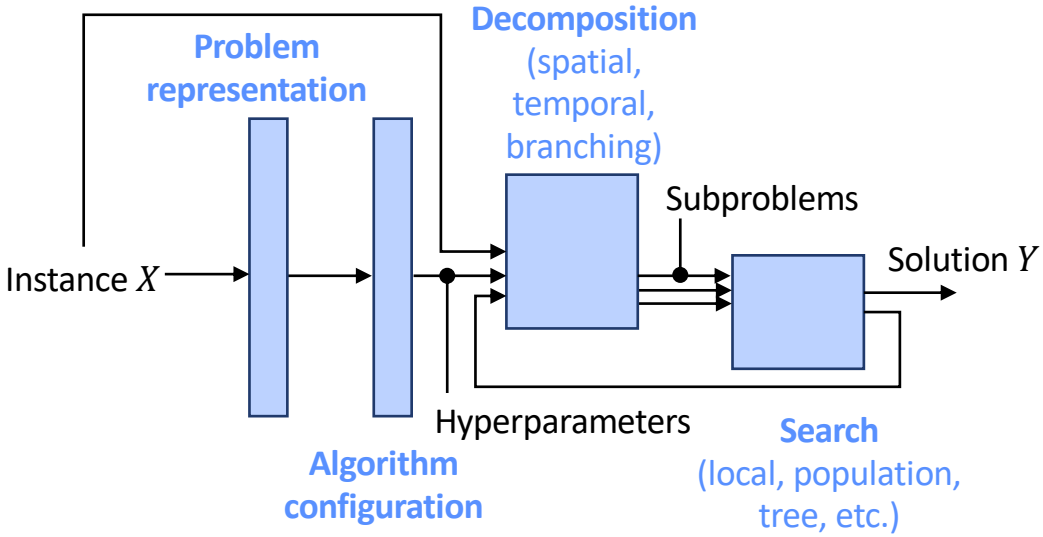


V. Jayawardana *et al.*, "Mitigating Metropolitan Carbon Emissions with Dynamic Eco-driving at Scale," *Transportation Research Part C: Emerging Technologies*, Apr. 2025.

Outline

1. Why is domain knowledge important for RL?
2. Strategies to mitigate RL sensitivity
3. Reward shaping
4. Imitation learning
5. Residual policy learning
6. **Learning-guided optimization**

Learning-guided optimization



- Definition: A **solver** is a composition of decision operators

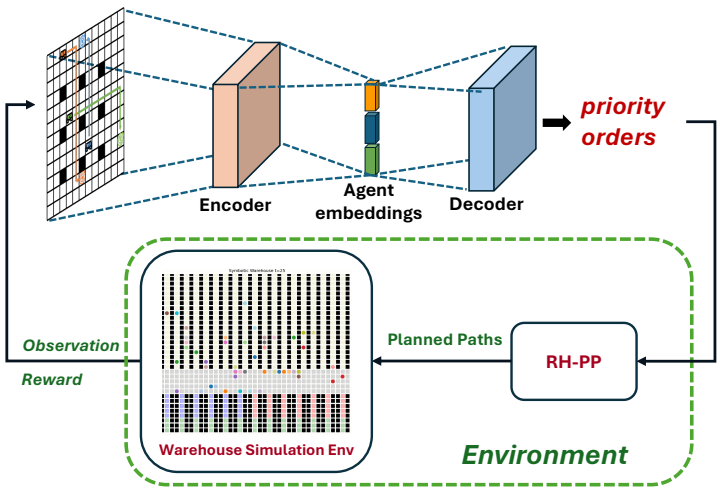
$$f: X \xrightarrow{o_1 \circ o_2 \circ \cdots \circ o_N} Y$$

  - Problem instance X, solution Y, operators $O \coloneqq \{o_j\}_{j \in N}$
  - Ex: Gurobi for integer programming, LKH-3 for routing, Prioritized Planning for multi-agent path finding
- Definition: A **learning-guided optimization** method is a solver with any ML operators,

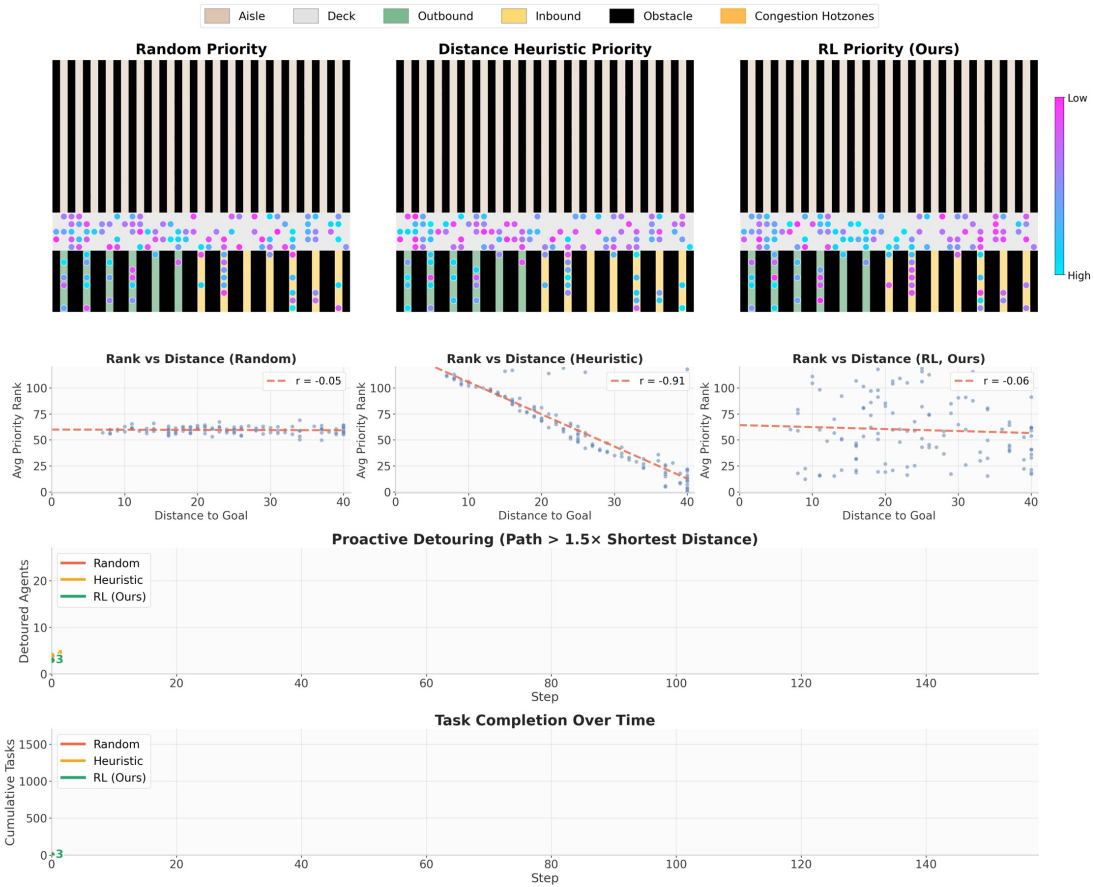
$$O_{\mathrm{ML}} \coloneqq \{o_i \mid o_i = o_{\theta_i}\} \neq \emptyset$$

Congestion Mitigation in Automated Warehousing

- Learn **priority order** of agents
- Method: Deep RL-guided prioritized planning
- Result: +20% throughput



symbotic



H. Zheng, Y. Ma, B. Araki, J. Chen, and **C. Wu**, "Learning-guided Prioritized Planning for Lifelong Multi-Agent Path Finding in Warehouse Automation," *Journal of Artificial Intelligence Research (JAIR)*, 2026.