How to Specify Reinforcement Learning Objectives

W. Bradley Knox
bradknox@cs.utexas.edu
University of Texas at Austin

James MacGlashan
james.macglashan@sony.com
Sony AI

Abstract

We discuss how practically to specify reinforcement learning (RL) objectives through careful design of reward functions and discounting. We specifically focus on defining a *human-aligned* objective for the RL problem, and we argue that reward shaping and decreasing discounting, if desired, are part of the RL solution—not the problem—and should be saved for a second step after this paper's focus. We provide tools for diagnosing misalignment in RL objectives, such as finding preference mismatches between the RL objective and human judgments and examining the indifference point between risky and safe trajectory lotteries. We discuss common pitfalls that can lead to misalignment, including naive reward shaping, trial-and-error reward tuning, and improper handling of discount factors. We also sketch candidate best practices for designing interpretable, aligned RL objectives and discuss open problems that hinder the design of aligned RL objectives in practice.

1 Introduction

Academic and practical treatments of reinforcement learning tend to assume that RL problem objectives are given. Yet in practice it is often not, and the RL objective—defined by a reward function and temporal discounting—must be designed by a human RL practitioner. This paper focuses on how to design an *aligned* RL objective, meaning one that provides an evaluative signal for what we humans desire from the RL agent. This RL objective is problem-side, independent of what RL solution is chosen, and its corresponding reward function is identified sometimes as *environmental* reward or *true* reward because it is defined independently of what specific RL algorithm is used to learn from it. Best practices for designing such RL objectives (before any reward shaping) have not yet been proposed. We provide a first attempt at collecting and synthesizing a succinct set of best practices for designing and debugging such *aligned* RL objectives *in practice.* In line with this focus on providing practical advice, this paper first considers how to find misalignment in an RL objective (Section 2) and three common sources of misalignment (Sections 3–5) before proposing how to design aligned RL objectives (Section 6).

1.1 Reward, return, and expected return

In the RL framework, a *reward function* specifies the immediate scalar reward received by the agent for taking action a_t in state s_t at time step t and transitioning to state s_{t+1}: $R(s_t, a_t, s_{t+1})$. A trajectory τ is a sequence of states and actions that either reaches an absorbing state (via episode termination) or is infinite. The *return* for some τ is its total discounted cumulative reward, $G(\tau) = \sum_{t=0}^{|\tau|-1} \gamma^t R(s_t, a_t, s_{t+1})$, where the discount factor γ reduces the value of future reward. The goal of an RL algorithm is to learn a policy π that maximizes the *expected return* across a distribution over start states $J(\pi) = \mathbb{E}_{\tau\sim\pi}[G(\tau)]$, where the expectation is taken over trajectories generated by following the policy π. We encourage the *precise* application of the terms reward, return, and expected return, because we suspect their common confusion—particularly using "reward" in place of the other terms—is itself a cause of poor literacy regarding the design of RL objectives and hinders the return-centric perspective for which we advocate.

1.2 A human-aligned RL objective

An RL objective is considered *aligned* if its reward function and discounting induces the same preference ordering over policies as the human stakeholders. A preference ordering over policies maps to a preference ordering over probability distributions over trajectories. In other words, trajectory distributions and policies that are preferred by humans should also achieve higher expected return according to the reward function and discounting. See Bowling et al. (2023) for a detailed theoretical treatment of the relationship between reward functions, discounting, and preferences over trajectory distributions, building upon Von Neumann and Morgenstern's expected utility theory (1944).[1] Roughly speaking, misalignment occurs when the RL objective incentivizes behaviors that are undesirable or even harmful from the human perspective.

An illustrative example To illustrate this concept of alignment, imagine a simple autonomous driving task with only 3 possible trajectories, illustrated in Figure 1. τ_{success} is a successful drive to its destination. In τ_{idle}, the ego vehicle remains parked and does not attempt to drive. And in τ_{crash}, the ego vehicle drives halfway to the destination and then crashes.

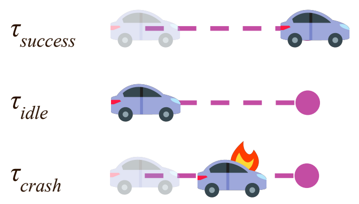

Figure 1: 3 abstract trajectories.

People generally prefer these trajectories in the order: τ_{success}, τ_{idle}, and τ_{crash}. An aligned RL objective would assign returns such that $G(\tau_{\text{success}}) > G(\tau_{\text{idle}}) > G(\tau_{\text{crash}})$, thereby inducing the same preference ordering. For instance, consider the following return arising from the reward function and discounting: $G(\tau_{\text{success}}) = 10$, $G(\tau_{\text{idle}}) = 0$, and $G(\tau_{\text{crash}}) = -50$. Such an RL objective is aligned in this simple task because the ordering of the trajectories by their returns matches the human preference ordering.

To broaden this example to include distributions over trajectories, now consider adding a fourth possibility, a so-called lottery that represents the risk inherent in any drive. Here the ego vehicle follows a policy that has a 90% chance of resulting in τ_{success} and a 10% chance of resulting in τ_{crash}. People tend to prefer the safe option τ_{idle} over the lottery, with its 1 in 10 chance of collision. A reward function and discounting that result in the previous set of returns would result in an expected return for this lottery of 4, which is *higher* than the return for τ_{idle}, indicating that such an RL objective would be *misaligned* in this slightly more complex world that incorporates risk. Practically speaking, this choice of RL objective incentivizes undesirably risky behavior over the preferred safe behavior.

Change your perspective from player to game designer The common perspective taken by an RL practitioner is that of the learning agent—an entity that pursues reward, estimates expected returns, and updates its policy to maximize expected return. However, when designing an RL objective, instead take the perspective of one who designs the rules of the game, who creates the scoring system that motivates player decisions. For RL, the scoring system is the calculation of return.

From this return-centric perspective, the choice of RL objective creates an ordering over policies based on their expected returns. The role of the designer is to craft a reward function and discounting that induce the desired ordering—one that aligns with human preferences about which outcomes (and distributions over outcomes) are better or worse. Getting this ordering right is critical, since the RL agent will pursue whatever behavior leads to the highest expected return under the RL objective it is given.[2]

[1] Bowling et al. rely on a generalization of the more common form of discounting we shared in the previous subsection, allowing their results to apply to tasks that are episodic and therefore undiscounted or that are continuing and either exponentially discounted or average reward.

[2] When designing RL objectives, the pursuit of alignment and ease of learning might appear at odds. We address this tension in Section 3. In short, rewards given to improve ease of learning should exclusively come from a separate reward shaping function, removing this tension by separating these two pursuits, with the added benefit of not overloading a reward function with two types of information. Additionally, there are many other ways to ease learning besides reward shaping.

1.3 Is RL the right approach for your sequential decision-making problem?

Although this work focuses on specifying RL objectives, RL is not the only approach to building sequential decision-making agents. Alternatives include *learning from demonstrations* (i.e., imitation learning) and *learning from preferences* (e.g., RL from human feedback, or RLHF). Many variants of these methods involve learning an intermediate reward model that is then used with RL to derive a policy. In such cases, much of this work's content is still relevant.

When deciding between designing an RL problem and learning from human input, several factors should be considered: the quality of human input that can be obtained (e.g., satisfactorily performant demonstrations); the cost of gathering human input (e.g., if a human must monitor the agent for safety, as is often the case for autonomous driving, then the cost of gathering input from that human might be small); the likelihood that one can tune RL to learn a satisfactory policy; and the cost of gathering experience for RL, especially when the learnt policy is not yet satisfactory. Additionally, we emphasize that RL can be used in conjunction with learning from human input. Including demonstrations, preferences, or other feedback can powerfully improve the ability of an RL algorithm to learn with less reward shaping (Knox & Stone, 2012; Brys et al., 2015), which may reduce incentives to create misaligned RL objectives.

Is RL potent enough? While RL has shown impressive results in certain domains (Silver et al., 2016; Wurman et al., 2022; Vinyals et al., 2019; Jumper et al., 2021), its important role in the training of the most impactful models today (e.g., LLMs) via RLHF (Ziegler et al., 2019; Ouyang et al., 2022; OpenAI, 2022; Glaese et al., 2022; Bai et al., 2022; Touvron et al., 2023) is merely to find multidimensional actions (of many tokens) that maximize reward in a bandit task, which falls short of the vision of RL as a maximizer of expected return over a horizon (Knox et al., 2024). RL generally appears to be a more difficult optimization problem than supervised learning from demonstrations and preferences—and therefore broadly effective RL algorithms may take longer to develop—but RL generally has a higher ceiling of performance *with well-designed RL objectives*, above what a human can demonstrate or differentiate via preferences.

Is RL safe enough? Additionally, the AI community is currently debating whether optimizing RL objectives can be sufficiently safe as the capabilities of AI increase (Cohen et al., 2024). This work—without taking sides on the debate—both considers the pitfalls of specifying RL objectives and ways to address them, attempting to clarify aspects of RL safety while making RL safer to deploy.

2 Finding misalignment in an RL objective

In our experience, one of the most effective ways to identify misalignment is to compare the simple preference orderings of human stakeholders to the orderings created by the RL objective.

Find mismatches in preference orderings This method seeks trajectories where the ordering implied by the RL objective contradicts the preferences of human stakeholders.

For example, consider the trajectories τ_{idle} (vehicle remains parked) and τ_{crash} (vehicle drives halfway and crashes). Humans would clearly prefer τ_{idle} (i.e., $\tau_{\text{idle}} \succ \tau_{\text{crash}}$). But a naive reward function that provides a seemingly large-magnitude collision penalty—let us say -100—and a small positive reward for speed—let us say it averages +1—*at every time step* might assign a higher return to τ_{crash} if it runs for more than 100 time steps before crashing. Knox et al. (2021) studied published reward functions that were designed for autonomous driving and found that 7 out of 9 reward functions incorrectly preferred crashing over not driving at all. This simple test revealed misalignment that could cause deadly behaviors if a real autonomous vehicle was blindly deployed after learning effectively from one of these reward functions.

We recommend two methods to find trajectories to compare: (1) choose desirable and undesirable trajectories that represent salient categories of different outcomes in the task, using your imagination or observations of policies' rollouts; and (2) attempt to do specification gaming yourself, mentally finding trajectories that accrue returns that contrast with their desirability. Note that the trajectories can be abstracted to only contain the information that the reward function measures, like speed and collisions in the example above, which makes them easier to compare and reason about otherwise.

Find undesired risk tolerance via indifference points Another technique for identifying misalignment is to compare a safe trajectory to a trajectory lottery, representing risk. For example, we can find the indifference probability p at which the reward function assigns equal expected return to:

- Option A: The safe trajectory τ_{idle}.
- Option B: A lottery that yields the successful trajectory τ_{success} with probability p and the crash trajectory τ_{crash} with probability $1-p$.

By examining the indifference probability p, we can assess whether the reward function exhibits an acceptable level of risk tolerance. In the same autonomous driving study, Knox et al. (2021) found that even the most risk-averse reward function was indifferent between not driving and a lottery with a crash rate more than 4000 times higher than that of a drunk US teenage driver. This second method of focusing on risk tolerance was broader than the first, showing that all 9 reward functions overly tolerate risk.

3 Misalignment by reward shaping

For current RL methods, *some* form of human guidance often appears needed. Reward shaping refers to the practice of guiding the learning process with human expertise by adding extra rewards to the base objective: $r_{shaped} = r(s_t, a_t, s_{t+1}) + r_{shaping}(s_t, a_t, s_{t+1})$, where the RL algorithm typically only experiences r_{shaped} as its reward. For example, in a navigation task, we might add intermediate rewards for progress towards the goal, rather than just a sparse reward upon reaching the goal. Reward shaping is predominant.For example, Knox et al. (2021) found that 13 of 19 reward functions used some form of reward shaping. Additionally, their study and our own experience unfortunately suggest that RL practitioners typically shape without maintaining a separate reward shaping function.

While well-intentioned, naive reward shaping often leads to misalignment between the shaped reward function and the true objectives of the task. The agent may learn to optimize for the shaping rewards in ways that are detrimental to the overall goal. For example, Randløv & Alstrøm (1998) appear to be the first to report a common mistake in reward shaping—rewarding progress without penalizing regress—which often causes agents to learn loops of progress followed by regress near the start state. Such looping navigation is incentivized by positive cumulative rewards per loop that under low discounting can add to large returns. In another example, Jaritz et al. (2018) designed reward functions for a racing video game that include a shaping penalty for deviating from the center of the lane. (See Knox et al. (2021, Appendix A.6) for full descriptions of their designed reward functions.) Discussing the misalignment caused by their shaping, Jaritz et al. write "the bots do not achieve optimal trajectories ... [in part because] the car will always try to remain in the track center."

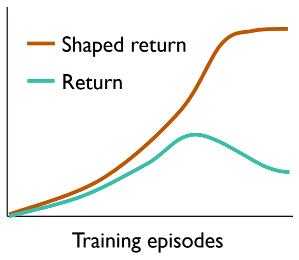

Figure 2: Illustration of using a separate shaping rewards function to catch shaped reward hacking after shaping helps during early learning.

There are some reward shaping methods that are guaranteed to preserve alignment (i.e., the ordering over policies), such as potential-based shaping (Ng et al., 1999). However, these methods are safe only under oft-ignored assumptions, and they appear to be used in a minority of instances in practice. See Appendix A for a discussion of the limitations of potential-based shaping.

We recommend to first design an aligned reward function, R, that directly captures human preferences over trajectory distributions, without any shaping. Then, if one wants to impart task knowledge via shaping—rather than by other means such as demonstrations[3]—do so a separate shaping reward function, $r_{shaping}$. This allows for clearer analysis of how the shaping affects learning and helps avoid

[3]In RL's standard text, Sutton & Barto (2018) encourage such other means over reward shaping: "... the reward signal is not the place to impart the agent prior knowledge about *how* to achieve what we want it to do... The reward signal is your way of communicating to the robot *what* you want it to achieve, not *how* you want it achieved... Better

overloading the base reward with both the true objective and potentially misaligned shaping rewards. One such analysis involves plotting both shaped and unshaped returns in learning curves, which will reveal when the RL algorithm is successfully optimizing for shaped expected return without benefiting—perhaps even harming—its accrual of aligned expected return, as Figure 2. (This phenomenon is often called *reward hacking*, which falsely implies that the agent knows the aligned objective and is willfully exploiting the shaped objective, thereby making the issue with the RL objective appear to be an issue of an malicious RL agent.)

4 Misalignment by trial-and-error reward design

Another common practice that can lead to misalignment is trial-and-error reward design. This iterative process typically involves this loop, starting with an initial candidate reward function:

1. Train an RL agent using the candidate reward function.
2. Observe the learned behavior of the agent during and after training.
3. If the result of learning are unsatisfactory, manually tweak the reward function—and perhaps tweak the RL algorithm too—and repeat from step 2.

Trial-and-error reward design is also predominant. Booth et al. (2023) found that 92% of surveyed RL experts used it to design their most recent reward function. Further, 100% of authors who shared their design process with Knox et al. (2021) used trial-and-error reward design.

Despite its intuitive appeal, trial-and-error reward design is problematic, as Booth et al. (2023) explain. First, the RL practitioner effectively acts as an optimization algorithm (aprocryphally dubbed "graduate student descent" by David McAllester) that can overfit the reward function to the specific training environment and RL algorithm. The resultant reward function may be brittle, unable to generalize well to other environments or algorithms, leading to misaligned behavior in new settings. Further, comparing an RL algorithm and its overfit reward function to another RL algorithm unfairly biases results towards the first algorithm and is a potential source of scientific invalidity in the RL literature, a concern that is yet uninvestigated. Second, this ad hoc tuning tends to produce RL objectives that are complex and thus hard to interpret. Such complexity hinders reasoning about what is actually being incentivized and whether it truly captures what humans desire.

Booth et al. (2023) asked RL experts to choose RL algorithms and reward functions for a simple grid world task called Hungry Thirsty (Singh et al., 2009) with a communicated evaluation metric. The subjects did engage in trial-and-error reward design: even in this time-constrained setting, subjects tried 4.1 reward functions on average and 93% tried more than one. Unsurprisingly, shaping was prevalent amidst trial-and-error design: 97% shaped their final reward function, despite that any of the available RL algorithms are able to learn an optimal policy with the simple reward function that is aligned with the evaluation metric. Overfitting to the finally chosen RL algorithm was widely observed.

A common cause of reward misdesign was using reward to rank the immediate desirability of states, such as giving high positive reward for reaching a goal state and low positive reward for reaching a subgoal. Critically, they often did not realize that smaller positive rewards could be achieved so frequently that their cumulative effect on return incentivizes the agent to avoid the larger rewards that are required for optimal behavior under the true evaluation function. They lacked the return-centric perspective we recommend in Section 1.2, focusing myopically on states rather than trajectories and return.

We recommend that any *iterative* design of the RL objective should focus on improving alignment, not performance. In other words, the reward function should be designed to represent human preferences over trajectory distributions, not just to produce appealing behaviors in a specific training setup.

places for imparting this kind of prior knowledge are the initial policy or initial value function, or in influences on these."

5 Misalignment by discounting

Temporal discounting is a key component of the RL framework (see Section 1.1). In the most common discounting regime—called exponential or geometric discounting—the discount factor $\gamma \in [0, 1]$ controls the relative weighting of immediate and future rewards. A reward k steps into the future is discounted by a factor of γ^k compared to an immediate reward.

In practice, RL implementations often use two different discount factors:

- γ_{task}: The true discount factor that is part of the underlying problem definition, helping to create returns that order trajectory distributions in an aligned way.
- γ_{alg}: The discount factor *hyperparameter* used by the RL algorithm during training.

Ideally, we would always set $\gamma_{\text{alg}} = \gamma_{\text{task}}$. Unfortunately, larger γ values in deep RL often lead to less stable learning. One cause for this instability is that most deep RL methods use *value bootstrapping*, meaning that the target for the value function update is defined to include an estimate from the learned value function. For example, in TD-learning (Sutton & Barto, 2018) the target for the value function at state s_t is set to $r_t + \gamma V_\theta(s_{t+1})$, where V_θ is the same deep value function network being trained. The effect of bootstrapping is increased with larger γ values since it increases the contribution of the value function estimate in the target. In general, there are no guarantees that value bootstrapping with function approximation is stable (Baird, 1995; Gordon, 1995; Van Hasselt et al., 2018; Fellows et al., 2023); values may diverge to infinity or grow exponentially. Whereas various works analyze conditions that affect stability and methods to address those conditions (Baird, 1995; Gordon, 1995; Sutton et al., 2008; Van Hasselt et al., 2018; Mnih et al., 2015; Fujimoto et al., 2018; Fellows et al., 2023; Gallici et al., 2024), in practice it remains hard to scale these methods to larger γ values. Consequently, most work still sets $\gamma_{\text{alg}} < \gamma_{\text{task}}$.

Given that typically $\gamma_{\text{alg}} \neq \gamma_{task}$, it is important to keep these two discount factors separate and to use γ_{task} when assessing the alignment of the reward function, including when plotting the true return such as in Figure 2.

γ	**10%**	**1%**	**0.1%**
0.9	2.3 s	4.6 s	6.9 s
0.99	23 s	46 s	69 s
0.999	230 s	460 s	690 s

Table 1: *Time to X% value* for future reward under different discount factors, assuming a 100 ms timestep.

To investigate why an RL agent is not pursuing a large future reward, one can compute the *time to X%* value—the number of timesteps in the future at which reward's contribution to the return is discounted to X% of its original value. *timesteps to* $X\% = log_{\gamma_{alg}} X \times 100$. If you know or can estimate the time per time step, then finish conversion to a unit of time. For example, with $\gamma = 0.9$ and a timestep of 100 ms, a reward's value is reduced to 10% after just 2.3 seconds. Even with $\gamma = 0.99$, the 10% threshold is reached after only 23 seconds.

With common γ_{alg} values and timesteps—such as 0.99 and 100 ms—events merely tens of seconds away may have little impact on what decisions the RL agent should take. A misaligned γ_{alg} can lead to shortsighted behaviors that prioritize near-term rewards over achieving highly desirable or undesirable outcomes in the long-term.

Applying our lens of seeking an aligned RL objective, the practical tactic above can be viewed as another form of assessing the trajectory ordering created by an RL objective. Orderings over trajectories—and more generally over trajectory distributions—are a valuable tool for evaluating both γ_{task} and γ_{alg}, providing a more precise version of the traditional advice to choose discounting to capture the desired trade-off between short-term and long-term outcomes.

6 How to design an aligned RL objective

We are unaware of any past proposals of best practices for the design of aligned reward functions and discounting. As a first draft of such best practices, we propose the following general process:

1. **Outcome variables** - Identify the minimum set of outcome variables that differentiate varying levels of success and failure. In other words, these outcomes are the high-level results that matter to

humans. In autonomous driving, these outcome variables might include whether the autonomous vehicle reaches its destination, if it avoids collisions, violations of traffic laws, passenger experience, and so on.

- For each outcome variable, define a per-timestep reward function component that, when summed over a trajectory, yields the total value of that outcome variable for the trajectory. For example, if one outcome variable is *whether the vehicle reaches its destination*, the corresponding per-timestep reward component could be a large positive value upon reaching the destination and zero everywhere else. For outcome variable *time taken to reach termination*, the corresponding per-timestep reward component could be -1, using a negative sign to indicate that less time is preferable.

2. **Representation** - Combine the per-timestep reward components into a reward function with parameters that are thus far undefined. As a strong default, we recommend a linear reward function with weights controlling the relative importance of each outcome variable. Only add complexity as needed, since simpler RL objectives are easier to tune and debug.
3. **Parameter tuning** - Tune the reward function parameters—e.g. the weights in a linear function—until the induced preference ordering over trajectory distributions aligns with human preferences. This step calibrates the contribution of each outcome variable to the overall RL objective.
4. **Evaluate and iterate** - Evaluate whether the RL objective adequately represents human preferences and, if not, iterate the process above. Critically, these iterations differ from the type of trial-and-error reward design described previously in that the purpose of iteration is to improve alignment, not performance.

In addition to further detail in Appendix C, we add two recommendations here.

- To ease reasoning about and debugging RL objectives, seek reward components and resultant returns that are interpretable to humans. For example, a binary indicator of goal achievement and a per-timestep time penalty are both highly interpretable, and their linear combination and undiscounted sum create returns that are likewise relatively interpretable.
- Mind the potential pitfalls discussed earlier, such as misalignment due to reward shaping, tuning via trial-and-error to directly improve RL performance, and overly myopic discounting. Avoiding these pitfalls can help prevent common sources of misalignment.

We emphasize that the process above is an incomplete sketch. Of particular note, we have largely overlooked discounting. In general, discounting should be viewed as part of the expressivity that defines the space of potential RL objectives; this space determines one's ability to find an RL objective that is aligned with any particular set of preferences. For episodic tasks, we recommend starting RL-objective design with a default of $\gamma_{task} = 1$, which is both common outside of deep learning settings and greatly eases reasoning about return. We refer the reader to Appendix B for discussion of discounting for continuing tasks and in general.

7 Conclusion

By following a principled process of designing reward functions and discounting—keeping alignment considerations at the forefront—we can make progress towards RL systems that behave in accordance with human values and preferences. However, despite our sketch above, no *validated* best practices exist and research is urgently needed as the effectiveness of optimizers for sequential decision making—including RL and planning algorithms—continue to improve.

Additionally, the recommendations in this paper sometimes require additional effort to follow, such as finding mismatches in preference orderings. More intuitive and accessible software tools should be developed to facilitate adherence to our our recommendations—as well as for holistically designing RL objectives—resulting in more aligned RL objectives.

By pursuing these and other open problems, the research community can work towards more reliably alignable RL systems that can be safely deployed in a wider range of real-world tasks. However, we must also remain vigilant to the risks and limitations of RL, and be willing to consider alternative paradigms when appropriate.

Acknowledgments

This work has taken place in part in the Rewarding Lab at UT Austin. The Rewarding Lab is supported by UT Austin's Good Systems grand challenge, ONR, and NSF.

References

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine learning proceedings 1995*, pp. 30–37. Elsevier, 1995.

Serena Booth, W Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 5920–5929, 2023.

Michael Bowling, John D Martin, David Abel, and Will Dabney. Settling the reward hypothesis. In *International Conference on Machine Learning*, pp. 3003–3020. PMLR, 2023.

Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

Noe Casas. Deep deterministic policy gradient for urban traffic light control. *arXiv preprint arXiv:1703.09035*, 2017.

Michael K Cohen, Noam Kolt, Yoshua Bengio, Gillian K Hadfield, and Stuart Russell. Regulating advanced artificial agents. *Science*, 384(6691):36–38, 2024.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Mattie Fellows, Matthew JA Smith, and Shimon Whiteson. Why target networks stabilise temporal difference methods. In *International Conference on Machine Learning*. PMLR, 2023.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning. *arXiv preprint arXiv:2407.04811*, 2024.

Amelia Glaese, Nat McAleese, Maja Trebacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. Improving alignment of dialogue agents via targeted human judgements. *arXiv preprint arXiv:2209.14375*, 2022.

Geoffrey J Gordon. Stable function approximation in dynamic programming. In *Machine learning proceedings 1995*, pp. 261–268. Elsevier, 1995.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Maximilian Jaritz, Raoul De Charette, Marin Toromanoff, Etienne Perot, and Fawzi Nashashibi. End-to-end race driving with deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2070–2075. IEEE, 2018.

Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, pp. 1181–1189, 2015.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

W. Bradley Knox and Peter Stone. Reinforcement learning with human and MDP reward. In *11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. IFAAMAS, June 2012.

W Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. Reward (mis)design for autonomous driving. *arXiv preprint arXiv:2104.13906*, 2021.

W Bradley Knox, Stephane Hatgis-Kessell, Sigurdur Orn Adalgeirsson, Serena Booth, Anca Dragan, Peter Stone, and Scott Niekum. Learning optimal advantage from preferences and mistaking it for reward. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 10066–10073, 2024.

Xiaoteng Ma, Xiaohang Tang, Li Xia, Jun Yang, and Qianchuan Zhao. Average-reward reinforcement learning with trust region methods. *arXiv preprint arXiv:2106.03442*, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 1928–1937. PMLR, 2016.

Abhishek Naik, Roshan Shariff, Niko Yasui, Hengshuai Yao, and Richard S Sutton. Discounted reinforcement learning is not an optimization problem. *arXiv preprint arXiv:1910.02140*, 2019.

A.Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. *Sixteenth International Conference on Machine Learning (ICML)*, 1999.

OpenAI. Chatgpt: Optimizing language models for dialogue. OpenAI Blog https://openai.com/blog/chatgpt/, 2022. Accessed: 2022-12-20.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Fifteenth International Conference on Machine Learning (ICML)*, pp. 463–471. Citeseer, 1998.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pp. 2601–2606. Cognitive Science Society, 2009.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Richard S Sutton, Csaba Szepesvári, and Hamid Reza Maei. A convergent o (n) algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in neural information processing systems*, 21(21):1609–1616, 2008.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton university press, 1944.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

Yiming Zhang and Keith W Ross. On-policy deep reinforcement learning for the average-reward criterion. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12535–12545. PMLR, 18–24 Jul 2021.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A On the impracticality of potential-based reward shaping for contemporary RL

Potential-based reward shaping—which we shorten to *potential shaping*—is a well-known reward shaping method that does not distort the true preference ordering of policies for discounted finite-state MDPs and undiscounted finite-state MDPs in which all policies eventually terminate.[4], making it a form of so-called *safe* reward shaping. Despite its renown, we found it to be rarely applied in prior work. Further, we have heard colleagues report negative results from trying it.

A potential reason for its lack of adoption in recent applications is that the same property that makes potential shaping safe in certain settings, is a property that may make it impractical for modern RL methods that use N-step returns or experience replay. To explain, we review potential shaping. Potential shaping starts with the original MDP reward function, $R(s, a, s')$. It then defines a new shaped reward function

$$\hat{R}(s, a, s') \triangleq R(s, a, s') + \gamma\phi(s') - \phi(s),$$

where $\phi(s)$ is a potential function that, loosely speaking, assigns a heuristic value to how good it is to be in any given state. Intuitively, if the agent transitions to better state s', then the difference $\gamma\phi(s') - \phi(s)$ will be positive, adding a positive value to the original reward that encourages the agent to take the action again. If it transitions to a worse state, a negative value will be added to the original reward that discourages the agent from taking the action again.

Potential shaping is safe under its assumptions because for any policy π, the potential-shaped Q-function, $\hat{Q}$, only differs from the original Q-function, Q, by a state-dependent offset: $\hat{Q}^\pi(s, a) = Q^\pi(s, a) - \phi(s)$. Since the offset is only state dependent, it does not affect the greedy action selection from each state.

Potential shaping results in this safe value function because the (discounted) potential value that is added from the next state, $\gamma\phi(s')$, is immediately canceled in the subsequent step. We can demonstrate this canceling by expanding the two-step discounted sum of shaped rewards:

$$\begin{aligned}
\hat{R}(s, a, s') + \gamma\hat{R}(s', a', s'') &= \Big(R(s, a, s') + \gamma\phi(s') - \phi(s)\Big) + \gamma\Big(R(s', a', s'') + \gamma\phi(s'') - \phi(s')\Big) \\
&= \Big(R(s, a, s') + \gamma\phi(s') - \phi(s)\Big) + \Big(\gamma R(s', a', s'') + \gamma^2\phi(s'') - \gamma\phi(s')\Big) \\
&= R(s, a, s') + \gamma\phi(s') - \phi(s) + \gamma R(s', a', s'') + \gamma^2\phi(s'') - \gamma\phi(s') \\
&= R(s, a, s') + \gamma R(s', a', s'') - \phi(s) + \gamma\cancel{\phi(s')} - \cancel{\gamma\phi(s')} + \gamma^2\phi(s'') \\
&= R(s, a, s') + \gamma R(s', a', s'') - \phi(s) + \gamma^2\phi(s'').
\end{aligned}$$

In general, the shaped N-step discounted return is:

$$\sum_{t=0}^{N-1} \gamma^t \hat{R}(s_t, a_t, s_{t+1}) = \sum_{t=0}^{N-1} \gamma^t R(s_t, a_t, s_{t+1}) + \gamma^N \phi(s_T) - \phi(s_0)$$

Originally, potential shaping was proposed for one-step online Q-learning methods. In this case, the subsequent removal of added potential values would not be immediately propagated to the estimated Q-value. Consequently, the first visit of a state-action pair would be biased toward good actions and biased away from bad actions. Only from many subsequent visits to the same state-action pair would its initial bias be diminished, converging to the bias-free potential-shaped value function $\hat{Q}^\pi$.

In contrast, modern on-policy RL methods such as PPO (Schulman et al., 2017), Impala (Espeholt et al., 2018), and A3C (Mnih et al., 2016), all use some form of large N-step returns that would

[4]Potential shaping is also safe for infinite-state MDPs under typical and mild constraints usually required for RL.

immediately cancel out the dense impact of the potential shaping,[5] leaving only a substantially discounted additional shaped reward based upon the final state of the N-step return: $\gamma^N \phi(s_N)$.

While modern off-policy RL methods such as SAC Haarnoja et al. (2018), TD3 Fujimoto et al. (2018), DDPG Casas (2017), and DQN Mnih et al. (2015) use single-step value updates, their use of experience replay may similarly result in fast cancellation of the dense potential values. That is, methods that use experience replay sample a large batch of state-action pairs and update all their values on each time step. Consequently, a single state-action pair may have its value adjusted many times before the agent visits it again in the environment, which can more quickly eliminate the bonuses from the shaping function. If methods pair experience replay with N-step returns, then the potential bonuses will be canceled even faster. Studying how fast these bonuses cancel in practice is a fruitful direction for future research and may lead to more effective and safe shaping approaches.

B Exponential discounting in continuing tasks

For episodic tasks, we recommend $\gamma_{\text{task}} = 1$. However, for continuing tasks, $\gamma_{\text{task}} = 1$ does not yield a meaningful objective because returns will accumulate to $\pm\infty$ for all but the most trivial reward function specifications. E.g., if the agent can accumulate a net positive amount of reward by the time it revisits a recurrent state in a continuing task, then as time progresses, the cumulative return will diverge to positive infinity. This divergence prevents a useful comparison of different policies and cannot capture a designer's preferences over policies.

There are two common solutions to this issue. Most common is to employ discounting with $\gamma < 1$. Another approach is to reframe the problem using an average reward objective, where the goal is to find a stationary policy[6] that maximizes the average reward. Unfortunately both approaches come with their own set of challenges, which we discuss below.

B.1 Exponential discounting

Although a task may be continuing in the sense that it lacks a clear terminating condition, in practice, agents are rarely run indefinitely. If we have an upper bound on the agent's runtime, we can utilize the "time to X" method discussed earlier to select a sufficiently large γ_{task} for evaluation purposes. Notably, it has been shown that every finite MDP has a critical point $\gamma^* < 1$, beyond which a *Blackwell optimal policy* emerges that is optimal for all $\gamma \in [\gamma^*, 1)$ Naik et al. (2019). While this critical point is specific to each MDP, it implies that if our upper bound exceeds γ^*, we have a path to optimality over even longer horizons.

Unfortunately, we cannot know in advance how big γ^* is. It could be much larger than we'd typically use in practice and choosing a $\gamma_{\text{alg}} < \gamma_{\text{task}}$ for a continuing task can complicate optimization. Specifically, when using $\gamma_{\text{alg}} < \gamma_{\text{task}}$ for stability reasons, we typically want the agent to consider all states it will encounter during its extended execution, rather than myopically focusing on the start state distribution. In tabular settings, we can achieve this by evaluating the discounted value of a policy at every state and *partially* ordering policies based on their dominance over others. Tabular methods enable finding an optimal policy within this partial ordering. However, when using function approximation, we need to define an objective function that can *totally* order policies to enable policy improvement.

A natural approach to mitigate overly myopic behavior when using function approximation is to define an objective function that weighs the discounted value function by the distribution of states induced by the policy: $J(\pi) = \sum_s d_\pi(s) V^\pi_\gamma(s)$, where $d_\pi(s)$ is the stationary probability that the agent visits state s when following policy π. However, standard discounted RL methods that greedily improve policies locally for individual states do not actually optimize this objective (Naik et al., 2019).

[5]These methods often use λ-returns which are more biased then Monte Carlo returns. While λ-returns would not as aggressively cancel potential shaping rewards as Monte Carlo returns, the shaping would nonetheless be greatly diminished compared to one-step updates. As $\lambda \to 1$, the less the method will benefit from potential shaping.

[6]A stationary policy is policy that does not change its probability distribution over actions for a given state. That is, if the agent revisits a state, the probability distribution will remain the same as the last time the agent visited the state.

Fortunately, this objective is equivalent to maximizing the average reward, and using average reward RL methods will directly optimize it, providing a more effective approach.

B.2 Average-reward objectives

Although choosing $\gamma_{task} = 1$ for a continuing task is problematic, continuing tasks can be undiscounted through the average reward RL objective. In this framework, the objective function is defined as: $J(\pi) = \sum_s d_\pi(s)\mathbb{E}_{a,s' \sim \pi(\cdot|s)}[R(s,a,s')]$. Here $J(\pi)$ is the average reward under the stationary distribution of policy π.

The average reward objective has several useful properties.

- Empirical evaluation is straightforward: simply average the reward signals over a sufficiently long trajectory (or set of trajectories) when evaluating an agent.
- The policy ordering it induces is equivalent to the policy ordering obtained when taking the limit of discounted objectives as γ approaches 1. Thus an optimal policy is defined by $\arg\max_\pi J(\pi) = \arg\max_\pi \lim_{\gamma \to \infty}(1-\gamma)V^{\pi,\gamma}(s_0)$, where s_0 is a start state and $V^{\pi,\gamma}$ is the value function under discount factor γ (Naik et al., 2019).
- Average reward deep RL algorithms avoid the mismatch between evaluation criteria (γ_{task}) and algorithm training parameters (γ_{alg}).

These properties make a strong case for using average-reward RL for continuing tasks. However, while there are emerging methods to solve average reward RL objectives (Ma et al., 2021; Zhang & Ross, 2021), methods have historically been difficult to use in practice and newer methods have not reached widespread adoption suggesting they may not yet be practical. As a result, discounted RL algorithms may still be the preferred algorithmic method, even with their optimization complications, until more practical algorithms for average reward RL are developed.

B.3 Reasons to use discounting in continuing-tasks

While we have primarily discussed the drawbacks of overly myopic discounted values with $\gamma_{\text{alg}} < \gamma_{\text{task}}$ and their potential for misalignment, there are scenarios where selecting a smaller γ may be beneficial in preventing misalignment.

A common concern regarding the safety of RL systems is that a long-lived, continuing-task planning agent may overoptimize a misaligned reward function, leading to perverse outcomes in the long term. However, using a smaller discount factor can mitigate this issue by restricting the space of policies that could be optimal for a given state space, action space, and reward function Jiang et al. (2015). This property was initially explored in the context of using discount factors to regularize the policies of model-based agents that plan with a learned model of the environment from limited data. However, a reduced policy space also facilitates easier evaluation of an agent's safety and inherently mitigates long-term perverse optimization of misaligned reward functions.

Therefore, if there is a high risk of reward function misalignment in a model-based, continual-task agent, discounting may be the preferred approach. Nevertheless, we still recommend evaluating agents with γ_{task} or the average reward to ensure that misalignment is not caused by excessive myopia.

C Designing an aligned RL objective

Here we add detail to the sketch of candidate best practices given in Section 6.

Outcome variables

Identifying the right outcome variables can be challenging, leading to a lengthy iteration process. To address this challenge, one strategy is to measure many more related outcome variables than initially anticipated. For example, in a project on autonomous racing, Wurman et al. (2022) measured numerous outcome variables. These outcomes included different types of collisions and passing events, which were measured to assess racing etiquette, allowing the reward function to trade off between this etiquette and and competitiveness. Although this set of outcome variable may be broader than those in the finally chosen RL objective, it can help indicate when fine-grained agent behavior differs from designer expectations.

Parameter tuning

Note that this step is effectively RLHF. However, standard RLHF appears to create highly shaped reward functions (Knox et al., 2024), which is not the aim here. Further, a linear reward function can be learned with far fewer samples than the shaped, deep reward functions common in contemporary RLHF. Because of this sample efficiency, a promising approach is to algorithmically identify preference inconsistencies and have the human designer rectify them, rather than assuming error in the form of a Boltzmann distribution, an assumption many RLHF researchers begrudgingly use, waiting for an improved model of error.

Evaluation

Since the usage of preferences in the parameter-tuning step already represents a sort of evaluation, what evaluation(s) would be helpful at this step is difficult to say without testing this design process broadly for real-world sequential domains. One potential source of informal additional evaluation is running RL on the designed objective, which might reveal that some important outcomes are not yet captured by the reward function. Another tactic is to actively search for trajectories for which the RL objective produces unintended incentives.