Breaking Free from Hand-Crafted Rewards: A Genetic Programming Framework for End-Goal-Driven Reinforcement Learning

Kamalesh Kumar
Computer Science
University of Massachusetts Amherst
Amherst, MA, United States
kkosalaraman@umass.edu

Jean-Alexis Delamer
Computer Science
St. Francis Xavier University
Antigonish, NS, Canada
jdelamer@stfx.ca

James Alexander Hughes
Computer Science
St. Francis Xavier University
Antigonish, NS, Canada
jhughes@stfx.ca

***Abstract*—Reinforcement Learning (RL) enables the training of autonomous agents to accomplish various tasks through environmental feedback. This feedback, formalized as rewards, guides the agent toward optimal strategies for achieving desired objectives. However, a fundamental challenge exists: agents optimize for reward maximization rather than directly solving the intended tasks. The reward functions, typically designed manually, do not necessarily lead to optimal solutions and may fail to capture the true complexity of the problem domain. Even reward functions that appear intuitive to human designers often produce suboptimal or unexpected agent behaviors.**

We introduce RevolveRL, a novel approach leveraging Genetic Programming to discover effective reward functions that enhance performance and reduce training time. Our method employs task-specific fitness functions designed around the ultimate goals of the tasks, which guide the evolutionary search process. We evaluate our approach on both classic RL benchmarks and the more complex MuJoCo physics simulation tasks. Our experimental results demonstrate that automatically evolved reward functions consistently yield superior policies and accelerate the training process compared to baseline manually designed rewards.

***Index Terms*—End-Goal-Driven Fitness; Genetic Programming; Reinforcement Learning; Reward Function.**

I. Introduction

Reinforcement Learning (RL) is a paradigm in machine learning used to train agents to complete specific tasks. RL leverages feedback mechanisms, often referred to as rewards, to drive the training process. The fundamental principle of RL involves an agent learning to make sequential decisions by interacting with an environment, where the objective is to maximize expected cumulative rewards over time [1].

A critical aspect of RL is that agents do not directly optimize for the intended goal itself; rather, they optimize for the reward signals they receive. This distinction highlights one of the key challenges in RL: designing reward functions that effectively guide agents toward accomplishing desired goals. Reward design becomes the primary interface through which developers communicate task objectives to learning agents [2].

However, manually engineered reward functions come with significant limitations. They require considerable domain expertise, are often vulnerable to exploitation by agents finding unintended shortcuts, and, perhaps most importantly, do not guarantee aligning with the target task. Even carefully crafted reward functions can lead to behaviors that maximize rewards without actually solving the intended problem [3], leading to reward hacking [4]. Reward design by humans is considered myopic, leading to unintended reward function overfitting and misdesign [5].

To address these limitations, we propose leveraging Genetic Programming (GP) for discovering effective reward functions. This paper presents RevolveRL, a novel framework that employs GP to search through the space of possible reward functions, identifying those that lead to superior agent performance and/or reduced training time. Our approach uses fitness functions explicitly designed around the end-goals of tasks rather than intermediate heuristics, ensuring alignment between evolved rewards and desired outcomes. Since each GP-evolved reward function must be evaluated by fully training an RL policy, the resulting fitness landscape becomes extremely noisy, non-stationary, and computationally expensive to navigate — conditions under which GP excels.

We evaluate our method across a set of RL environments, including classic control problems and the more complex MuJoCo physics simulation tasks. Our experiments demonstrate that GP-evolved reward functions can significantly outperform manually designed alternatives, both in terms of final policy quality and sample efficiency during training.

From an evolutionary computation perspective, our contribution lies not only in applying GP to reward design, but in demonstrating an effective evolutionary computation formulation for optimizing reward functions in large-scale RL training loops.

II. Related Work

The problem of reward function search has been studied before in multiple contexts. Early works by [6] and [7] introduced the concept of an optimal reward function and positioned it within an evolutionary framework. They illustrate that there can exist many reward functions that can improve agent learning performance, and the speed at which it converges to

the optimal solution. The most similar line of work to ours is an early one by [8], where the authors propose a GP-based search for reward functions. The key difference is that they propose to only find adjustments to an existing fitness-based reward function.

The extended optimal reward problem introduced in [9] tackles some of the limitations in [8] by finding Pareto optimal solutions of a multi-objective optimization problem that favors reward functions with best behavior and least learning time. Although their formalism is amenable to single and multi-agent settings, the experimental results are limited to grid-world and traffic assignment scenarios, as is the case with [8]. It is also unclear if their results hold when extended to the Deep RL setting, and when using recent policy learning algorithms instead of Q-Learning.

Another closely related work is by [10], which addresses sparse rewards using symbolic regression to generate dense intrinsic rewards. Their method combines symbolic reward learning with evolutionary search over policy space, ultimately selecting the best policy based on fitness. However, their focus is on finding optimal policies rather than discovering suitable reward functions. In comparison to policies, reward functions entail more robust and transferable properties [11]. Unlike their approach, which uses the original reward as the fitness signal, our method employs an end-goal-driven fitness function tailored to the desired task outcome.

More broadly, reward function design is also addressed by reward shaping. Reward shaping consists of modifying the original reward using potential-based functions that encode domain knowledge [12], [13]. However, they require significant manual effort and tuning, especially when multiple shaping terms are involved [14]. Reward design has also been studied in more interesting avenues. For example, recent works have explored the idea of generating reward functions for compositionally specified tasks, where high-level task specifications are used to aid in the search [15], [16], [17].

Evolutionary algorithms in general have been used in various contexts in RL, which all fall under evolutionary reinforcement learning. We refer the reader to [14] for a detailed review on the same.

III. Problem Description

A. Reinforcement Learning

Reinforcement Learning problems are commonly modeled as a Markov Decision Process, defined by the tuple (S, A, P, R, γ), where:

- S is the set of states;
- A is the set of actions;
- $P(s'|s, a)$ is the transition probability from state s to s' given action a;
- $R(s, a)$ is the reward function;
- $\gamma \in [0, 1]$ is the discount factor.

The goal of RL is to find an optimal policy π^* that maximizes the expected total return:

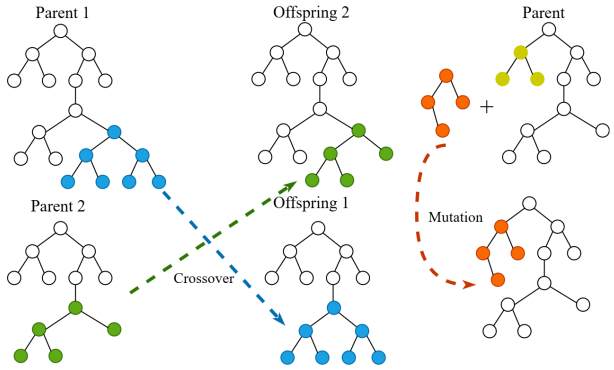


Fig. 1: Illustration of genetic programming operations: Mutation (right) replaces a randomly selected subtree, and crossover (left) exchanges subtrees between two syntax trees at the crossover points.

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right],$$

where the expectation $\mathbb{E}_{\pi}$ is taken over the trajectories induced by the policy π, with transitions sampled according to $s_t \sim P(\cdot|s_{t-1}, a_{t-1})$ and actions chosen as $a_t \sim \pi(\cdot|s_t)$.

To solve RL problems, methods are typically divided into value-based and policy-based approaches. Value-based methods, such as Q-learning, focus on estimating the value of each state or state-action pair. Policy-based methods, like Proximal Policy Optimization (PPO), optimize a parameterized policy directly to maximize expected rewards.

B. Genetic Programming

Genetic Programming (GP) is an evolutionary computation technique that evolves populations of programs to solve problems. Unlike traditional optimization methods that search through a fixed-dimensional space, GP typically explores the space of candidate programs represented as syntax trees.

In brief, GP operates on a population of candidate programs, each of which is a potential solution to the target problem. The fitness of a program ϕ is evaluated using a fitness function $F(\phi) \rightarrow \mathbb{R}$ that quantifies how well the program performs on the target task. Highly fit programs are selected to propagate to the subsequent generation and may be altered via genetic operators. The process continues for a fixed number of generations or until a termination criterion is met, with the best program representing the solution to the problem.

IV. RevolveRL

RevolveRL is a novel framework that leverages GP to discover effective reward functions for reinforcement learning tasks. The framework consists of two main components:

- **Reward Function Evolution**: An evolutionary algorithm that evolves a population of reward functions.
- **Policy Learning**: A policy learning algorithm that uses the evolved reward functions to train the agent.

A. End-Goal-Driven Fitness Functions

The fitness function, which guides the evolutionary process, is defined based on the underlying goal of the environment. For example, in a maze navigation task, the fitness function might capture the number of remaining steps toward the goal, so as to solve the maze as fast as possible. Prior works calculate the fitness of a reward function at the end of the learning phase, which is only indicative of the final performance of a policy that was optimized for a particular reward function.

For a given environment, we first define a performance metric $\mathcal{M}$ that directly measures the success of a policy in achieving the desired end-goal. This metric operates on the trajectory $\tau_\pi = \{(s_t, a_t)\}_{t=0}^{T}$ of state-action pairs generated by policy π, where s_t and a_t represent the state and action at time step t, and T is the episode length. The metric $\mathcal{M}$ could be task completion rate, average episode return, learning speed, or any other relevant measure of policy performance.

Formally, we then define the fitness function $F(\phi)$ for a reward function ϕ as:

$$F(\phi) = \mathcal{M}(\tau_{\pi_\phi}),$$

where π_ϕ is the policy learned using reward function ϕ. The optimality of a reward function ϕ^* is defined as:

$$\phi^* = \arg\max_{\phi \in \mathcal{P}} F(\phi),$$

where $\mathcal{P}$ is the space of possible reward functions. This formulation ensures that we are not merely maximizing rewards, but rather finding reward functions that lead to policies that accomplish the intended objectives.

In this work, we consider an effective reward function for a problem as the one that not just helps a policy to converge to the optimal solution, but also helps the agent to learn faster. Thus, the final fitness score of a reward function is calculated based on two components:

- F_{train}: the mean episodic fitness across all episodes during training, which captures learning efficiency and stability throughout the training process;
- F_{post}: the mean episodic fitness over n episodes using the learned policy, which measures final performance achieved with the evolved reward function.

The final fitness score $F_{\text{total}}(\phi_i)$ for a reward function ϕ_i is the sum of both components:

$$F_{\text{total}}(\phi_i) = F_{\text{train}}(\phi_i) + F_{\text{post}}(\phi_i)$$

This two-component design ensures that we select reward functions that not only lead to good final performance, but also enable efficient learning throughout the training process.

B. Genetic Programming Framework

Our GP implementation starts with a randomly initialized population of reward functions represented as syntax trees. The internal nodes represent primitives sampled from a pre-defined primitive set, while leaf nodes are either terminal constants or reward function arguments. Tree heights are constrained within minimum and maximum bounds, with equal probability of generating full or varying-depth trees.

A generational algorithm is used where elitism of one and tournament selection populates subsequent generations. Single-point crossover operator exchanging subtrees between candidate solutions and single-point mutation operator replacing a subtree are used as the genetic operators and are applied probabilistically.

Our implementation uses the DEAP framework [18] with the primitive set detailed in Table I.

=

TABLE I: Primitive functions used in our approach

Primitive	Input	Output
Addition (+)	`[float, float]`	`float`
Subtraction (−)	`[float, float]`	`float`
Division (/)	`[float, float]`	`float`
Multiplication (⋆)	`[float, float]`	`float`
Logical AND	`[bool, bool]`	`bool`
Logical OR	`[bool, bool]`	`bool`
Logical NOT	`[bool]`	`bool`
Equality (==)	`[float, float]`	`bool`
Less Than (<)	`[float, float]`	`bool`
Greater Than (>)	`[float, float]`	`bool`
If-then-else	`[bool, bool, bool]`	`bool`
If-then-else	`[bool, float, float]`	`float`

V. Experiments

To evaluate the effectiveness of our GP-based reward function discovery framework, we conducted experiments across a diverse set of environments. Our evaluation encompasses both continuous control tasks from the MuJoCo physics simulation suite [19] and discrete environments including Taxi [20] and BlackJack [1]. This diverse selection of environments allows us to assess the generality and robustness of our approach.

Beyond evaluating the overall performance of our framework, we also conduct an ablation study to validate our two-component fitness function design. Further, the baseline reward functions for control tasks in MuJoCo add certain penalties to limit torque/power consumed. We conduct additional analysis to show that the improved performance from our approach is not simply because of omitting such explicit handcrafted penalties. We also perform cross-algorithm validation using Deep Q-Network (DQN) on the Taxi environment to demonstrate that our evolved rewards provide benefits beyond specific algorithm-environment combinations.

A. Experimental Setup

In our approach, the policy learning component employs Proximal Policy Optimization (PPO) [21] as the base algorithm. For each environment, we conducted training over one million steps, with mean episodic fitness calculated from rollouts of 2048 steps. To ensure statistical significance, all results are averaged across 50 random seeds.

The GP parameters were tuned through preliminary experiments. We initialized a population of 50 individuals, with

crossover and mutation probabilities set to 0.8 and 0.1, respectively. Tournament selection was performed with a size of 5, and elitism of 1 was used to preserve the best solutions. The evolutionary process ran for 100 generations.

The population size and generation count were constrained by the high cost of fitness evaluation: each generation requires training 50 reward functions across 10 random seeds, with every seed involving a full 1-million-step PPO run. Fitness is computed as the average metric across these 10 seeds. This results in roughly 500 million environment steps per generation (50 billion over the full run), with wall-clock runtimes of 7–12 days depending on the environment.

B. Environments

1) MuJoCo Benchmark: Our primary evaluation focuses on eight MuJoCo environments presenting distinct continuous control challenges. These environments include forward movement tasks (Ant, HalfCheetah, Hopper, Humanoid Walking, Swimmer), balance tasks (Inverted Pendulum, Inverted Double Pendulum), and postural control (Humanoid Standup).

For each environment, we designed goal-oriented fitness functions that directly measure task success. For forward movement tasks, the fitness function measures the distance traveled in the forward direction, encouraging efficient locomotion. The Humanoid Standup task uses the mean z-coordinate of the torso as fitness, promoting upright posture. For inverted pendulum tasks, the fitness is based on the pole's orientation, with the goal of maintaining vertical alignment.

The hand-crafted reward functions used as a baseline comparison are the standard reward formulations commonly used in these environments. We used these to evaluate the effectiveness of our GP-evolved reward functions.

2) Additional Environments: To demonstrate the generality of our approach, we evaluated three additional environments with distinct characteristics: Bipedal Walker (continuous control requiring complex coordination), Taxi (discrete navigation task), and BlackJack (decision-making under uncertainty).

The fitness functions for these environments were designed to capture their specific objectives. For the Bipedal Walker, we employ a binary fitness function that rewards successful terrain crossing while penalizing failures. The Taxi environment maximizes the number of remaining steps to encourage efficient navigation and task completion. In the BlackJack environment, we use a binary fitness function that focuses on game outcome, rewarding wins and penalizing losses.

C. Comparative Baselines

To evaluate RevolveRL, we first benchmark it against the naïve approach, which directly uses PPO with the original reward function. Further, we also use the following diverse set of reward discovery/shaping methods as baselines: (a) LISR (Learning Inrinsic Symbolic Rewards, [10]), a symbolic reward-based intrinsic reward generator, and (b) EORP (Extended Optimal Reward Problem, [9]) that finds Pareto-optimal solutions by using Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [22]. For both of these approaches, the initial reward population is set as done in the respective works. Apart from these two baselines, we also include approaches based on reward shaping. More specifically, we include (c) ReLara (Reward Shaping for Reinforcement Learning with An Assistant Reward Agent, [23]), which introduces a dual-agent architecture where a secondary "reward agent" learns to construct dense auxiliary signals to guide the primary policy agent. Additionally, we also compare against ROSA (RL Optimizing Shaping Algorithm, [24]), which frames reward learning as a "game of two partners", optimizing the shaping reward through a cooperative game-theoretic interaction between the agent and the environment. We utilize the standard hyperparameter setting considered in the implementations of all the baselines in our comparison.

D. Evaluation Metrics

We evaluate the effectiveness of our approach using four key metrics: learning speed, final performance, training stability, and reward–task alignment. Learning speed is measured through the convergence rate and the number of steps required for the agent to reach stable performance. Final performance is assessed using the mean episodic fitness after convergence, reflecting the agent's ultimate task proficiency. Training stability is quantified by the variance of episodic fitness throughout training, indicating the consistency and reliability of the learning process.

To evaluate how well the reward functions capture the underlying task objective, we compute the *Task Alignment Coefficient* (TAC) introduced in [25]. TAC measures the Pearson correlation between (i) the cumulative returns obtained under a learned reward function and (ii) the corresponding *proxy task scores* (when human preferences across trajectories is unavailable), i.e., the environment's true fitness signals such as forward distance, success indicator, or balance height. Formally, for N evaluation episodes with learned-reward returns $\{r_i\}$ and proxy fitness scores $\{y_i\}$,

$$\mathrm{TAC} = \mathrm{corr}(r, y) \in [-1, 1].$$

A higher TAC indicates stronger reward–task alignment, meaning the reward function ranks behaviors in a manner consistent with the true task objective.

To ensure statistical rigor, we additionally perform permutation tests comparing the distributions of episodic fitness values between agents trained with GP-evolved rewards and those trained with the original reward function. This provides quantitative evidence of the statistical significance of observed performance differences.

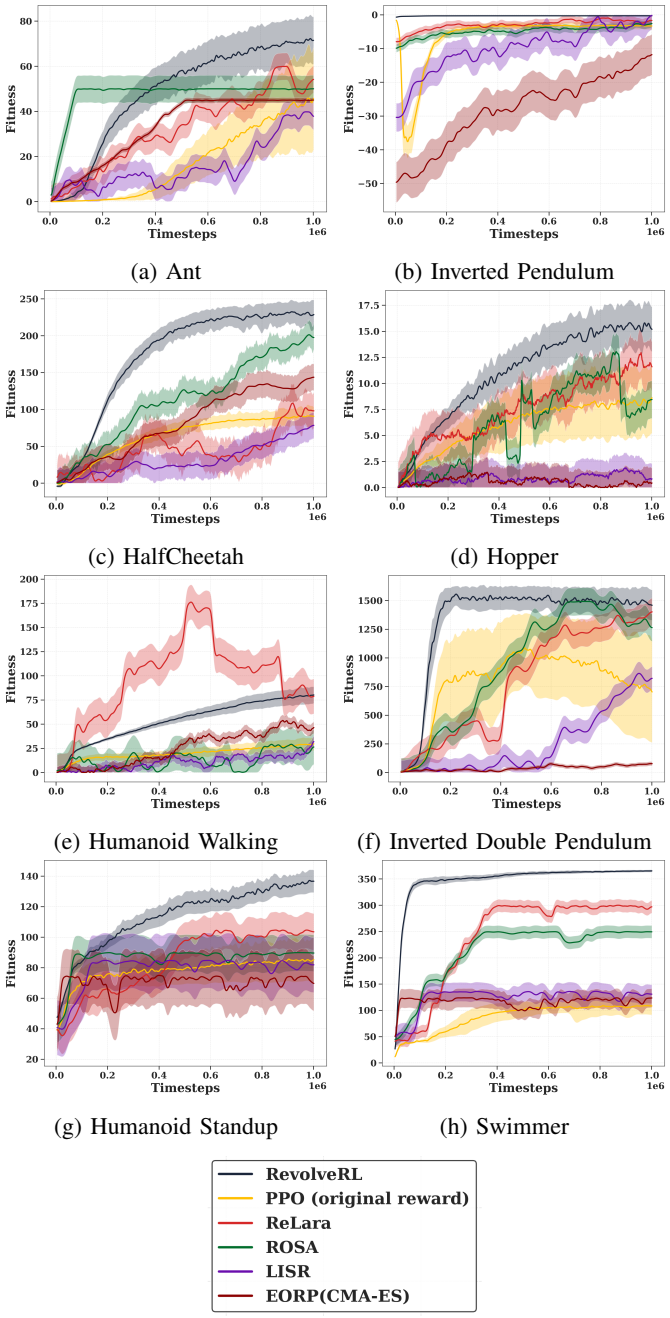
VI. Results

A. MuJoCo Benchmark Performance

In all eight environments, we observed that the GP-evolved reward functions consistently outperformed the baseline.

The key performance improvements include:

- **Faster Convergence**: GP-evolved reward functions led to more rapid learning, with agents reaching higher performance levels in fewer training steps;



(a) Ant (b) Inverted Pendulum

(c) HalfCheetah (d) Hopper

(e) Humanoid Walking (f) Inverted Double Pendulum

(g) Humanoid Standup (h) Swimmer

Fig. 2: Training performance comparison between RevolveRL, and other baselines across various continuous control tasks in the MuJoCo suite.

- **Higher Final Performance**: At the end of training, agents using evolved reward functions achieved significantly better goal-oriented performance metrics;
- **Training Stability**: The learning curves for GP-evolved rewards showed lower variance throughout training, indicating more consistent learning progress.

This improved performance was particularly notable in environments like Inverted Pendulum, where the GP-evolved reward function enabled a near-optimal policy that maintained the pole in an almost perfectly upright position throughout the episode, with near-zero variance in performance.

B. Performance on Additional Environments

For the additional environments, the GP-evolved reward functions significantly outperformed the original reward functions in all three cases.

The most striking result was observed in the Taxi environment, where the agent completely failed to learn using the original reward function, but showed rapid learning progress with the evolved reward function. The learning curve for the GP-based reward showed quick convergence to an optimal solution, while the original reward function resulted in essentially random behavior throughout training.

In the Bipedal Walker environment, the GP-evolved reward function led to more consistent successful terrain traversals compared to the original reward. Similarly, in BlackJack, the evolved reward function agent achieved higher winning rates.

These results demonstrate how our approach can transform previously challenging or unlearnable tasks into ones that can be efficiently solved through better reward design.

C. Evolved Reward Functions

The GP-evolved reward functions demonstrate stronger task alignment than hand-crafted rewards. Notable examples include: Taxi evolved to -1 per timestep (encouraging task completion speed), BlackJack uses the player's hand score directly, and Humanoid Standup reduces to a simple product of two state variables.

D. Statistical Significance & Task Alignment

From the permutation tests (with 10^5 permutations) on all environments, we obtained p-value $< 10^{-4}$, indicating that the performance improvements are statistically significant.

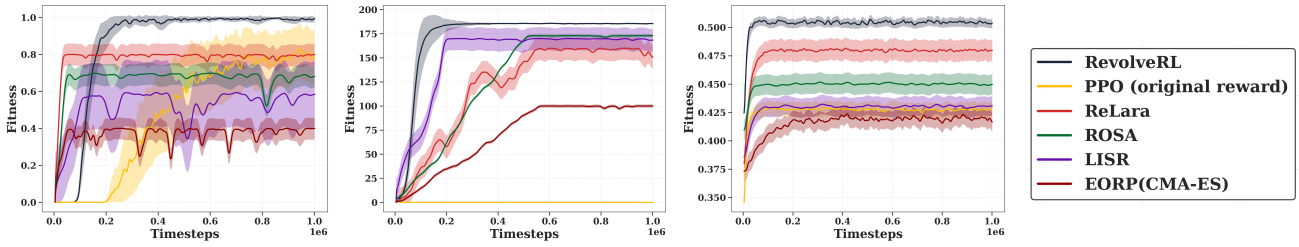
It's important to note that the reward functions generated by our GP algorithm after 100 generations are not necessarily optimal within the space of all possible reward functions. For environments where the agent's movement lacks an upper bound (such as forward-moving MuJoCo tasks), extending the evolutionary process over more generations would likely discover even more effective reward functions.

With the respect to the reward–task alignment, the TAC metric is summarized in Table II, and reward functions obtained from RevolveRL showcase higher task alignment via higher TAC values across majority of the environments tested.

E. Ablation Studies

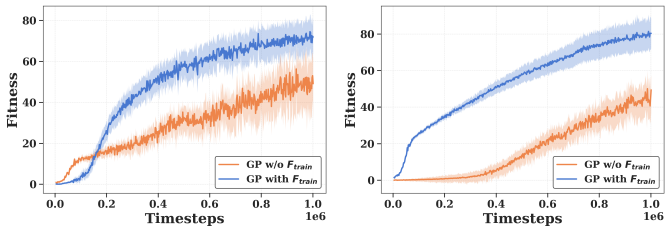
1) Impact of F_{train} Component: To validate the effectiveness of our two-component fitness function design, we conduct an ablation study focusing on the necessity of the F_{train} component. This study compares our full approach ($F_{\text{total}} = F_{\text{train}} + F_{\text{post}}$) against a variant that uses only post-training fitness ($F_{\text{total}} = F_{\text{post}}$).

The ablation study aims to answer whether incorporating training performance feedback during the evolutionary process



(a) Bipedal Walker (b) Taxi (c) BlackJack

Fig. 3: Training performance comparison between RevovleRL and other baselines for Bipedal Walker, Taxi, and BlackJack environments. The GP-evolved rewards show significant improvements in learning efficiency and final performance.



(a) Ant Environment (b) Humanoid Environment

Fig. 4: Ablation study results comparing the full approach with F_{train} (blue) versus using only F_{post} (orange).

TABLE II: Task Alignment Coefficient (TAC) across environments for different methods. The highest value for each environment is shown in **bold**. Higher is better.

Environment	RevolveRL	PPO (orig. reward)	ReLara	ROSA	LISR	EORP
Ant	**0.8927**	0.7224	0.7745	0.7412	0.7163	0.7342
Inverted Pendulum	**0.9321**	0.8351	0.8923	0.8886	0.8425	0.7603
HalfCheetah	**0.7865**	0.7032	0.7214	0.7583	0.6801	0.7407
Hopper	**0.8849**	0.7208	0.7989	0.7748	0.2157	0.1061
Humanoid Walking	0.6504	0.6153	**0.6706**	0.4445	0.3473	0.6202
Inverted Double Pendulum	0.8392	0.7014	0.8567	**0.8712**	0.6691	0.1184
Humanoid Standup	**0.5186**	0.2789	0.4484	0.3731	0.3324	0.1189
Swimmer	**0.7841**	0.3308	0.7278	0.6809	0.4198	0.3735
Bipedal Walker	**0.9532**	0.8157	0.9031	0.8752	0.7582	0.6881
Taxi	**0.9615**	0.0421	0.9256	0.9404	0.9314	0.7552
BlackJack	**0.8691**	0.7102	0.8220	0.7963	0.7336	0.6937

is essential for discovering effective reward functions, or if relying solely on final policy performance is sufficient. We tested both variants across two representative environments: Ant and Humanoid. The results demonstrate clear benefits of including the F_{train} component:

- **Faster Convergence**: The full approach with F_{train} consistently achieved faster convergence to high-quality reward functions during the evolutionary process. On average, the approach with F_{train} required 20-30% fewer generations to reach the same level of performance compared to using only F_{post}.
- **Better Final Performance**: The reward functions evolved with F_{train} achieved superior performance across both environments. In the Humanoid environment, the improvement was particularly pronounced, with a 15% increase in final episodic fitness compared to the ablated version.

The results indicate that F_{train} serves as a crucial component by providing intermediate feedback during policy learning, allowing the evolutionary process to discover reward functions that not only lead to good final policies but also facilitate efficient learning. Without this component, the evolutionary search relies solely on final performance, which may miss reward functions that provide better learning dynamics during training.

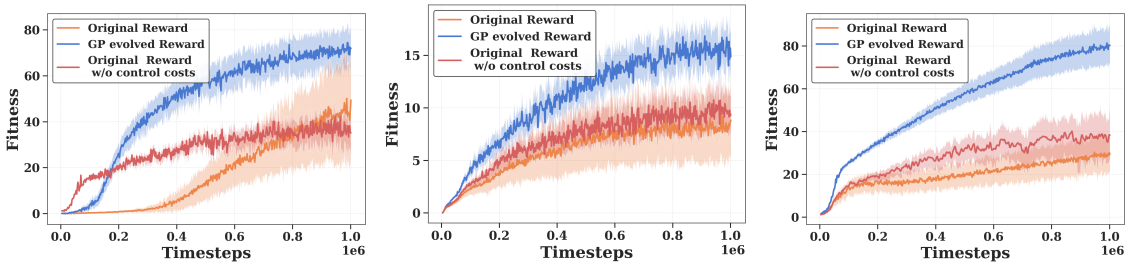
2) Comparison of Control Costs for tasks in MuJoCo: It may be noted that the original reward functions incorporated additional penalty terms, such as control and contact costs. These penalties, as mentioned in Gymnasium's documentation, are to penalize the agent from taking actions that are too large or having large contact forces. Since the fitness functions don't explicitly account for these penalty terms, it is important to analyze if the gain in performance is not merely from omitting these penalties. Figure 5 illustrates the case for 3 complex MuJoCo tasks where the agent was trained with the original reward function but omitting the penalty terms (both control & contact cost terms). From the figure, it is clear that this modified reward function, although slightly performs better than the baseline, does not outperform the evolved reward.

Further, we also analyzed the distribution of the costs via violin plots, and Figure 6 summarizes the result. It's obtained from a sample of 5000 episodes (as each of the 50 random seeds from training is evaluated for 100 episodes post-training). The y-axis represents the total penalty (which is indicative of the torque consumed by the motors) in an episode. It can be observed that the GP reward has roughly the same median compared to the original reward with/without penalty term (in Ant and Humanoid, the GP reward actually incurs lower costs). Hence, this confirms that the gain in performance from the evolved reward functions does not simply arise from neglecting the constraints on torque/power consumed, but rather could be attributed to the efficiency of the evolved reward functions themselves.

3) Case Study on the Taxi Environment: The Taxi environment provides a particularly illustrative example of how our approach can transform reward design. The original reward function assigns:

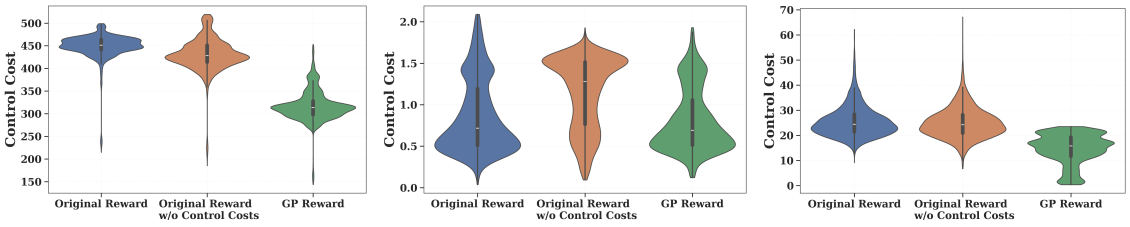
- +20 for the correct passenger drop-off;
- -10 for illegal actions;
- -1 otherwise.

While this reward function appears intuitive, it fails to



(a) Ant (b) Hopper (c) Humanoid

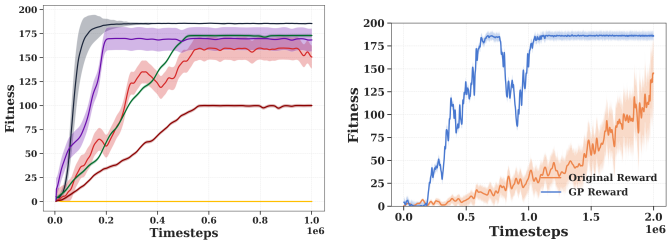
Fig. 5: Ablation study results on the original reward functions with and without the penalty terms. Excluding the penalty terms does not aid the original reward function to outperform the GP-evolved reward.



(a) Ant (b) Hopper (c) Humanoid

Fig. 6: Violin plot representing the distribution of the control costs. The GP-evolved reward incurs lower costs (via the median represented by the white dash) in Ant and Humanoid.

guide the learning process effectively when using PPO (under multiple hyperparameter settings) because the agent rarely attempts the drop action due to the potential large penalty from illegal actions. To verify that this is not simply a limitation of the PPO algorithm, we conducted additional experiments using DQN on the same environment with the original reward function (Fig. 7). Our results show that while DQN does learn with the original reward function, it fails to achieve near-optimal performance within 2,000,000 training steps.



(a) PPO with Original Reward and GP-Evolved Reward (b) DQN with Original Reward and GP-Evolved Reward

Fig. 7: Performance on Taxi for PPO and DQN. While DQN learns with the original reward, it fails to achieve near-optimal performance.

The GP search discovered a simpler reward function that assigns -1 at every time step, which drastically improves learning for PPO by incentivizing the agent to complete the task as quickly as possible to avoid accumulating penalties. When we tested this evolved reward function with DQN, it achieved near-optimal performance within 500,000 training steps, demonstrating that the evolved reward function enables effective learning across different algorithms and extends beyond specific algorithm-environment combinations.

VII. Discussion

RevolveRL's consistent superiority over the baselines in convergence speed and task alignment is primarily driven by its novel fitness formulation that explicitly co-optimizes for learning efficiency (F_{train}) and ground-truth alignment (F_{post}), a simple, yet effective strategy that is absent in methods like LISR or EORP that focus largely on final policy outcomes. Our ablation studies confirm that F_{train} acts as a crucial "teaching" signal by favoring rewards that provide dense, smooth feedback gradients early in training. Moreover, unlike reward shaping approaches (e.g. ReLara, ROSA) that rely on auxiliary agents or heuristics which can be exploited, RevolveRL evolves symbolic structures directly against the task objective. This results in superior TAC and the discovery of robust rules — such as the succinct penalty in the Taxi domain — that avoid the local minima and "reward hacking" pitfalls prevalent in hand-crafted designs. Finally, the evolutionary search naturally identifies energy-efficient control strategies without explicit constraints, demonstrating that the method captures the intrinsic physics of the task more effectively than manually tuning penalty coefficients.

VIII. Conclusion and Future Work

In this work we presented RevolveRL, a novel framework that leverages genetic programming to discover effective reward functions for reinforcement learning tasks. Our approach

differs from traditional reward shaping methods by focusing on end-goal-driven fitness functions and evolving complete reward functions rather than just adjusting existing ones.

Our results demonstrate that GP-evolved reward functions consistently outperform hand-crafted baselines across a range of environments, including continuous control tasks from MuJoCo and discrete environments like Taxi and BlackJack. We also evaluated RevolveRL across a diverse set of baselines, ranging from evolutionary approaches to automated reward-shaping techniques and showed the consistent task alignment of RevolveRL reward functions on majority of the benchmarks tested via the Task Alignment Coefficient. The evolved reward functions not only achieve better final performance, but also enable faster learning and more stable training processes. Further, we also showed that the evolved reward function are not only better performing, but also ensure that the constraints on power/energy are met, as required in control tasks in MuJoCo. Additionally, our algorithm-specific analysis in the Taxi environment reveals that while certain algorithm-reward combinations may fail to learn, our evolved rewards provide learning benefits across different RL algorithms.

Our key contributions include: (1) a novel GP-based framework for automatic reward function discovery, eliminating manual reward engineering; (2) an end-goal-driven fitness function design ensuring task alignment; (3) comprehensive evaluation across diverse environments and baselines with statistical validation and metrics such as the Task Alignment Coefficient, and (4) validation of our training-based fitness component and demonstration of algorithm-agnostic benefits.

Looking forward, several promising directions emerge for extending this work. We plan to investigate the application of our framework to multi-objective tasks and complex reward functions, which would broaden its applicability to real-world scenarios. Additionally, we aim to explore methods for improving the interpretability of evolved reward functions, which could provide valuable insights into effective reward design principles. Finally, the potential for transferring learned reward functions across similar environments presents another exciting avenue for research.

The code for this work is made available on github: https://github.com/jdelamer/RevolveRL

IX. Acknowledgments

This research was enabled in part by support provided by the Digital Research Alliance of Canada (alliancecan.ca).

References

[1] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[2] J. Eschmann, "Reward function design in reinforcement learning," *Reinforcement learning algorithms: Analysis and Applications*, pp. 25–33, 2021.

[3] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. Dragan, "Inverse reward design," *Advances in neural information processing systems*, vol. 30, 2017.

[4] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," *arXiv preprint arXiv:1606.06565*, 2016.

[5] S. Booth, W. B. Knox, J. Shah, S. Niekum, P. Stone, and A. 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*, vol. 37, no. 5, 2023, pp. 5920–5929.

[6] S. Singh, R. L. Lewis, and A. G. Barto, "Where do rewards come from," in *Proceedings of the annual conference of the cognitive science society*. Cognitive Science Society, 2009, pp. 2601–2606.

[7] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg, "Intrinsically motivated reinforcement learning: An evolutionary perspective," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 70–82, 2010.

[8] S. Niekum, A. G. Barto, and L. Spector, "Genetic programming for reward function search," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 83–90, 2010.

[9] R. Grunitzki, B. C. da Silva, and L. A. Bazzan, "Towards designing optimal reward functions in multi-agent reinforcement learning problems," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[10] H. U. Sheikh, S. Khadka, S. Miret, S. Majumdar, and M. Phielipp, "Learning intrinsic symbolic rewards in reinforcement learning," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[11] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278—287.

[12] B. Marthi, "Automatic shaping and decomposition of reward functions," in *Proceedings of the 24th International Conference on Machine learning*, 2007, pp. 601–608.

[13] M. Grzes and D. Kudenko, "Learning potential for reward shaping in reinforcement learning with tile coding," in *Proceedings AAMAS 2008 Workshop on Adaptive and Learning Agents and Multi-Agent Systems (ALAMAS-ALAg 2008)*, 2008, pp. 17–23.

[14] H. Bai, R. Cheng, and Y. Jin, "Evolutionary reinforcement learning: A survey," *Intelligent Computing*, vol. 2, p. 0025, 2023.

[15] K. Jothimurugan, S. Bansal, O. Bastani, and R. Alur, "Compositional reinforcement learning from logical specifications," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 026–10 039, 2021.

[16] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2107–2116.

[17] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 73, pp. 173–208, 2022.

[18] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "Deap: Evolutionary algorithms made easy," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 2171–2175, 2012.

[19] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.

[20] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[22] C. Igel, N. Hansen, and S. Roth, "Covariance matrix adaptation for multi-objective optimization," *Evolutionary computation*, vol. 15, no. 1, pp. 1–28, 2007.

[23] H. Ma, K. Sima, T. V. Vo, D. Fu, and T.-Y. Leong, "Reward shaping for reinforcement learning with an assistant reward agent," in *Forty-first international conference on machine learning*, 2024.

[24] D. Mguni, T. Jafferjee, J. Wang, N. Perez-Nieves, W. Song, F. Tong, M. Taylor, T. Yang, Z. Dai, H. Chen *et al.*, "Learning to shape rewards using a game of two partners," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, 2023, pp. 11 604–11 612.

[25] C. Muslimani, K. Johnstonbaugh, S. Chandramouli, S. Booth, W. B. Knox, and M. E. Taylor, "Towards improving reward design in rl: A reward alignment metric for rl practitioners," *arXiv preprint arXiv:2503.05996*, 2025.