Self-Refined Large Language Model as Automated Reward Function Designer for Deep Reinforcement Learning in Robotics

Yang Li
PhD student
University of Manchester
yang.li-4@Manchester.ac.uk

MANCHESTER 1824
The University of Manchester

Introduction

- The paper proposes a novel LLM framework with **a self-refinement mechanism** for **automated reward function design.**
- Current Reward Function Design:
  - Meticulous Manual Crafting reward design
  - AutoRL [1]: automate the hyperparameters and reward function tuning
    - predefined, parameterized reward function and subsequently fine-tune its parameters to identify an optimal reward function like using evolutionary algorithms [2]
    - due to its dependency on an initially hand-crafted parameterized reward function, AutoRL lacks the ability to formulate a reward function entirely from scratch

Current methods rely on domain-specific expertise !!!

[1] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, et al. 2022. Automated reinforcement learning (autorl): A survey and open problems. Journal of Artificial Intelligence Research 74 (2022), 517–568.
[2] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In AAAI Conference on Artificial Intelligence, Vol. 33. 4780–4789.

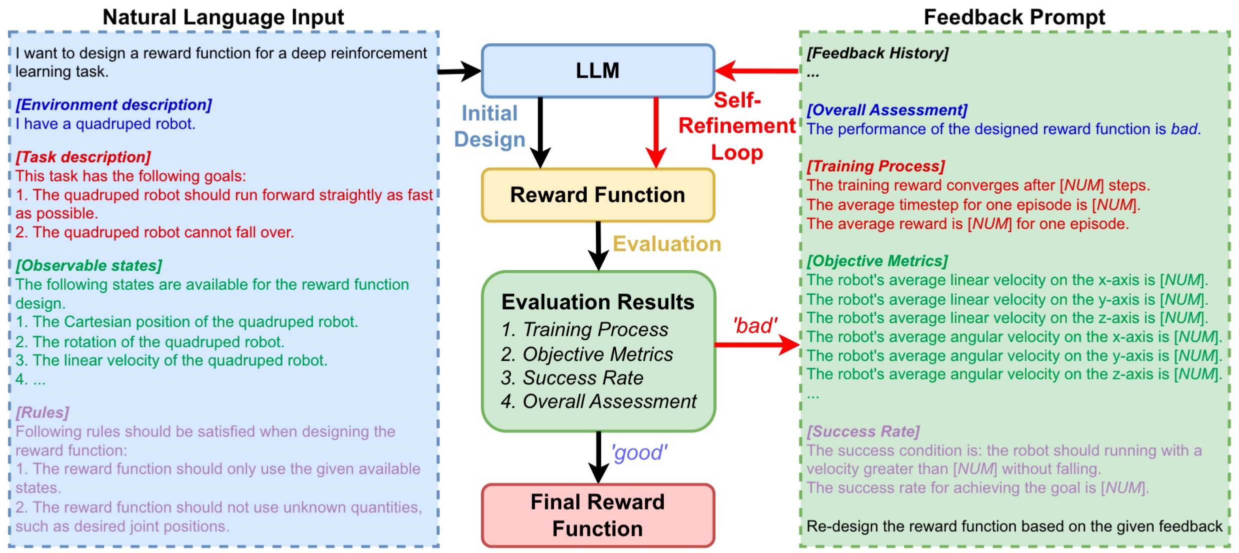
Introduction

- The **common-sense knowledge** of LLM offers the potential to ***alleviate*** the human effort required in formulating reward functions
- Current Reward Function Design with LLM:
  - For simpler tasks, such as normal-form games, LLM could serve directly as a proxy reward function [1].
  - Through processing natural language instructions, LLM seamlessly integrates task requirements and user preferences into reward functions [2].

LLM is able to independently design a reward function from scratch for continuous robotic control tasks?

[1] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. 2023. Reward design with language models. arXiv preprint arXiv:2303.00001 (2023).
[2] Hengyuan Hu and Dorsa Sadigh. 2023. Language instructed reinforcement learning for human-ai coordination. arXiv preprint arXiv:2304.07297 (2023).

Method



I want to design a reward function for a deep reinforcement learning task.

[Environment description]
I have a quadruped robot.

[Task description]
This task has the following goals:
1. The quadruped robot should run forward straightly as fast as possible.
2. The quadruped robot cannot fall over.

[Observable states]
The following states are available for the reward function design.
1. The Cartesian position of the quadruped robot.
2. The rotation of the quadruped robot.
3. The linear velocity of the quadruped robot.
4. ...

[Rules]
Following rules should be satisfied when designing the reward function:
1. The reward function should only use the given available states.
2. The reward function should not use unknown quantities, such as desired joint positions.

[Feedback History]
...

[Overall Assessment]
The performance of the designed reward function is *bad*.

[Training Process]
The training reward converges after [*NUM*] steps.
The average timestep for one episode is [*NUM*].
The average reward is [*NUM*] for one episode.

[Objective Metrics]
The robot's average linear velocity on the x-axis is [*NUM*].
The robot's average linear velocity on the y-axis is [*NUM*].
The robot's average linear velocity on the z-axis is [*NUM*].
The robot's average angular velocity on the x-axis is [*NUM*].
The robot's average angular velocity on the y-axis is [*NUM*].
The robot's average angular velocity on the z-axis is [*NUM*].
...

[Success Rate]
The success condition is: the robot should running with a velocity greater than [*NUM*] without falling.
The success rate for achieving the goal is [*NUM*].

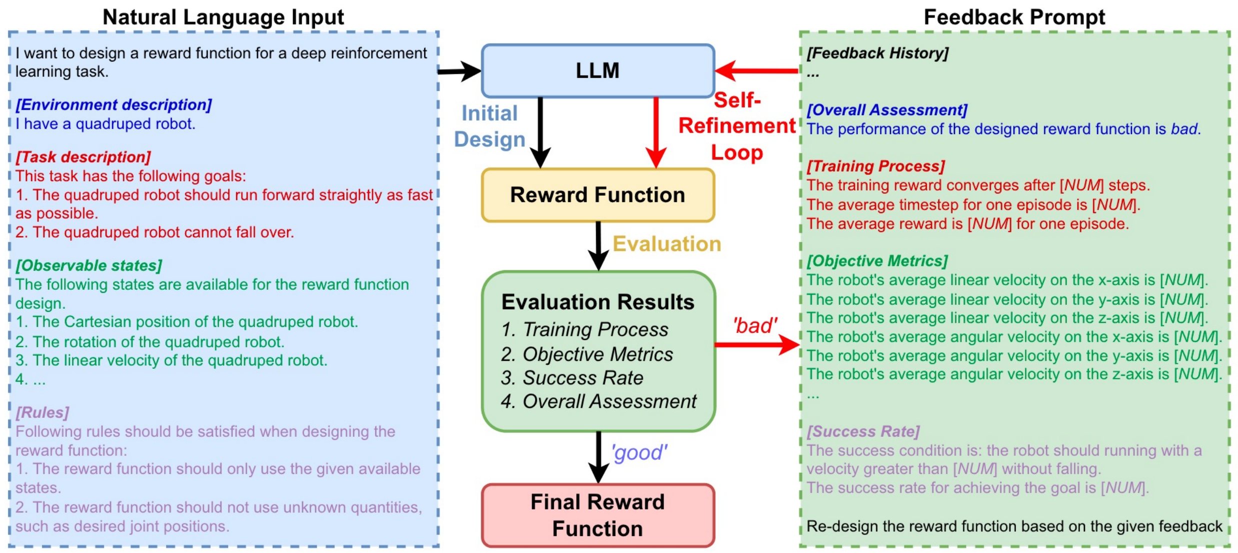
Re-design the reward function based on the given feedback

Figure 1: Our proposed self-refine LLM framework for reward function design. It consists of three steps: *initial design*, *evaluation*, and *self-refinement loop*. A quadruped robot forward running task is used as an example here. A complete list of the prompts used in this work can be found in the appendix.

The framework consists of three steps:

1) **Initial design,** where the LLM accepts a natural language instruction and devises an initial reward function;
2) **Evaluation,** where the system behavior resulting from the training process using the designed reward function is assessed;
3) **Self-refinement loop,** where the evaluation feedback is provided to the LLM, guiding it to iteratively refine the reward function.

Method



I want to design a reward function for a deep reinforcement learning task.

[Environment description]
I have a quadruped robot.

[Task description]
This task has the following goals:
1. The quadruped robot should run forward straightly as fast as possible.
2. The quadruped robot cannot fall over.

[Observable states]
The following states are available for the reward function design.
1. The Cartesian position of the quadruped robot.
2. The rotation of the quadruped robot.
3. The linear velocity of the quadruped robot.
4. ...

[Rules]
Following rules should be satisfied when designing the reward function:
1. The reward function should only use the given available states.
2. The reward function should not use unknown quantities, such as desired joint positions.

[Feedback History]
...

[Overall Assessment]
The performance of the designed reward function is *bad*.

[Training Process]
The training reward converges after [*NUM*] steps.
The average timestep for one episode is [*NUM*].
The average reward is [*NUM*] for one episode.

[Objective Metrics]
The robot's average linear velocity on the x-axis is [*NUM*].
The robot's average linear velocity on the y-axis is [*NUM*].
The robot's average linear velocity on the z-axis is [*NUM*].
The robot's average angular velocity on the x-axis is [*NUM*].
The robot's average angular velocity on the y-axis is [*NUM*].
The robot's average angular velocity on the z-axis is [*NUM*].
...

[Success Rate]
The success condition is: the robot should running with a velocity greater than [*NUM*] without falling.
The success rate for achieving the goal is [*NUM*].

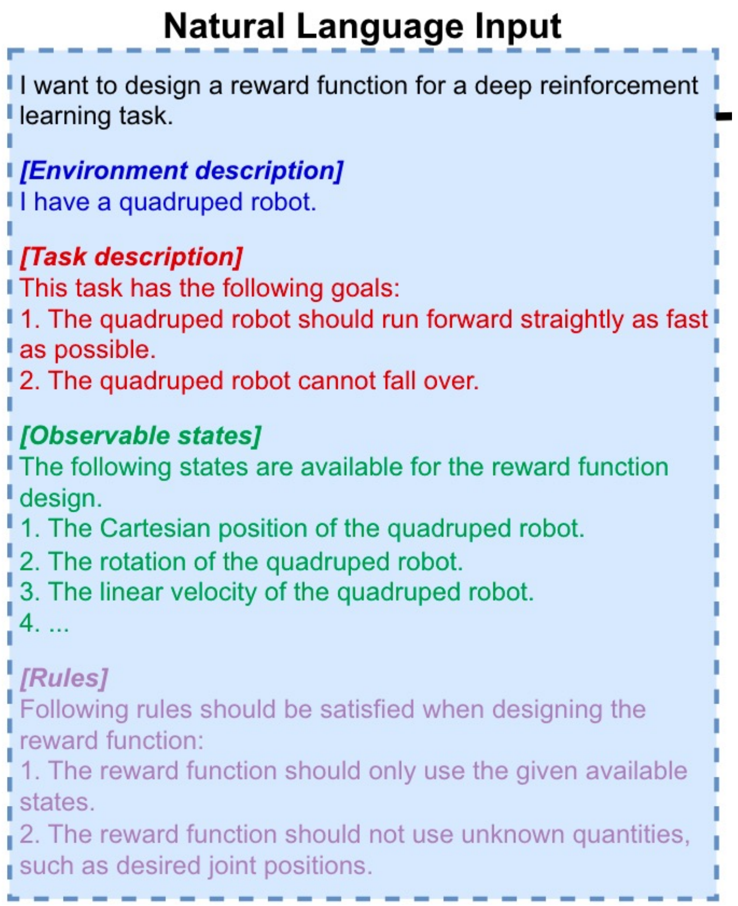
Re-design the reward function based on the given feedback

Figure 1: Our proposed self-refine LLM framework for reward function design. It consists of three steps: *initial design*, *evaluation*, and *self-refinement loop*. A quadruped robot forward running task is used as an example here. A complete list of the prompts used in this work can be found in the appendix.

Contributions:

1. We explore the ability of LLM to design reward functions for DRL controllers. Diverging from many studies that leverage few-shot in-context learning when prompting the LLM, we employ the LLM as a **zero-shot reward function designer.**
2. We incorporate a **self-refinement mechanism** into the reward function design process to enhance its outcomes.
3. We highlight the effectiveness and applicability of our proposed approach through **a variety of continuous robotic control tasks across diverse robotic systems.**

Method – Initial Design


Natural Language Input

I want to design a reward function for a deep reinforcement learning task.

[Environment description]
I have a quadruped robot.

[Task description]
This task has the following goals:
1. The quadruped robot should run forward straightly as fast as possible.
2. The quadruped robot cannot fall over.

[Observable states]
The following states are available for the reward function design.
1. The Cartesian position of the quadruped robot.
2. The rotation of the quadruped robot.
3. The linear velocity of the quadruped robot.
4. ...

[Rules]
Following rules should be satisfied when designing the reward function:
1. The reward function should only use the given available states.
2. The reward function should not use unknown quantities, such as desired joint positions.

They segment the natural language prompt into four parts:

- **Environment description:** we first describe the robotic system we are working with, e.g., a quadruped robot or a 7-DOF manipulator, and provide details regarding the environmental setup;
- **Task description:** we then outline the control objectives of the task, along with any existing specific task requirements;
- **Observable states:** we also provide a list of the observable states that are available for the reward function design;
- **Rules:** finally, we explain the rules that the LLM should follow when designing the reward function. Specifically, we emphasize two rules: first, **the reward function should be based solely on the observable states;** second, the **reward function should exclude elements that are challenging to quantify, such as specific target postures of the quadruped robot.**

Method – Initial Design

Prompt example:

Ball Catching

I want to design a reward function for a reinforcement learning task.

I have a 7 DOF manipulator with a gripper attached as the end effector. The manipulator initially holds a container that opens upwards, and a ball will be thrown from above.

This task has the following goals:
1. The manipulator should use the container to capture the ball before it falls on the ground.
2. When the ball has been captured, it should stay within the container as long as possible.

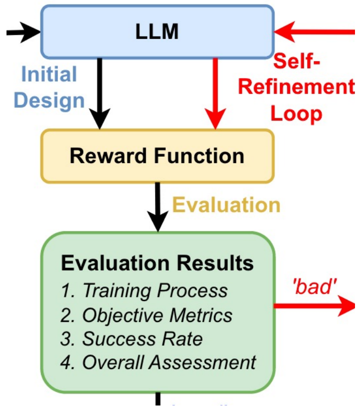
The following variables are available for reward function design.
1. ball_pos: This vector with a dimension of 3 represents the x,y,z position of the ball according to the world coordinate.
2. ball_vel: This vector contains the linear velocity of the ball according to the world coordinate.
3. container_pos: This vector represents the x,y,z position of the container according to the world coordinate.
4. container_rot: This 4-dimensional vector represents the rotation of the container in quaternion.

Some rules while designing the reward function:
1) The reward function should only use the given variables.
2) The reward function should not use unknown quantities, such as desired joint positions, in its computation.

Design a complete reward function for this task.

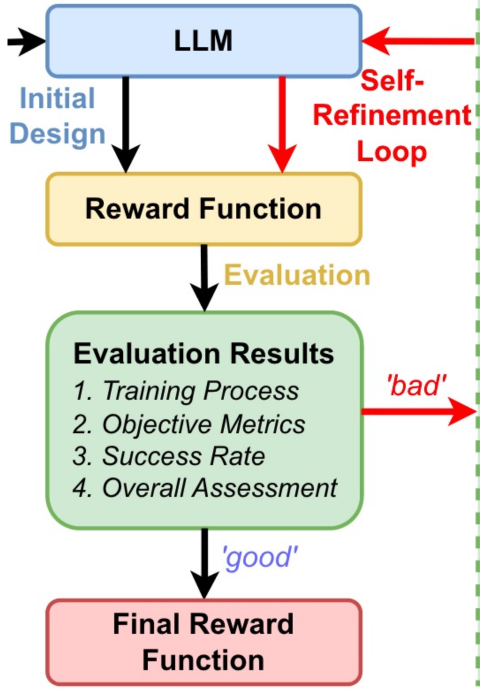
Method - Evaluation

Aiming to minimize human intervention, the evaluation is structured as an automated procedure.



- **Training process:** this summary includes information on whether the reward has converged, the average reward per training episode, and the average number of timesteps in each episode.
- **Objective metrics:** we then represent the overarching performance metric G(T) with multiple individual task-specific objective metrics gk(T), k = 1, . . . , ng. Each objective metric gk(T) addresses an aspect of the task requirements.
  - *For instance, in the quadruped robot's straight-forward walking task, two objective metrics could be employed: one assessing the forward distance the robot travels without toppling and another quantifying any unintended lateral movements. We then compute the average values of these objective metrics gk(T) over all sampled trajectories.*
- **Success rate in task accomplishments:** in addition to the task-specific objective metrics, we also introduce the success rate SR of the trained policy in accomplishing the designated control task as a general and task-agnostic criterion. For each control task, we define a success condition using Signal Temporal Logic (STL) to capture the core objective of the task.
  - *For example, the success condition for a quadruped robot walking task could be that the forward distance travelled without falling should exceed a predetermined threshold. A trajectory meeting the success condition is considered a success. The success rate SR is determined across all sampled trajectories.*

Method – Self-Refinement Loop



Feedback Prompt

[Feedback History]
...

[Overall Assessment]
The performance of the designed reward function is *bad*.

[Training Process]
The training reward converges after [*NUM*] steps.
The average timestep for one episode is [*NUM*].
The average reward is [*NUM*] for one episode.

[Objective Metrics]
The robot's average linear velocity on the x-axis is [*NUM*].
The robot's average linear velocity on the y-axis is [*NUM*].
The robot's average linear velocity on the z-axis is [*NUM*].
The robot's average angular velocity on the x-axis is [*NUM*].
The robot's average angular velocity on the y-axis is [*NUM*].
The robot's average angular velocity on the z-axis is [*NUM*].
...

[Success Rate]
The success condition is: the robot should running with a velocity greater than [*NUM*] without falling.
The success rate for achieving the goal is [*NUM*].

Re-design the reward function based on the given feedback

Training process

Objective metrics

} intrinsically task-dependent

Success rate } overall performance of the designed reward function R as either 'good' or 'bad'

Method – Self-Refinement Loop

Prompt example:

Ball Catching

The performance of the RL agent trained with the designed reward function is [*good*|*bad*].

The training reward converges after [*NUM*] steps.
The average timestep for one episode is [*NUM*].
The average reward is [*NUM*] for one episode.

The average normalized action value is [*NUM*]
The normalized distance between the ball and the container is [*NUM*]
The average distance between the ball and the container on the x-axis is [*NUM*]
The average distance between the ball and the container on the y-axis is [*NUM*]
The average distance between the ball and the container on the z-axis is [*NUM*]
The ball's average linear velocity on the x-axis is [*NUM*]
The ball's average linear velocity on the y-axis is [*NUM*]
The ball's average linear velocity on the z-axis is [*NUM*]

Evaluation results of [*NUM*] trials are given below:
Goal 1 success rate is [*NUM*]

Redesign the reward function based on the given feedback.

The University of Manchester

Experimental Results

- *Robotic manipulator (Franka Emika Panda Emika (2023))*:
  1. *Ball catching*: the manipulator needs to catch a ball that is thrown to it using a tool (Fig. 2a);
  2. *Ball balancing*: the manipulator should keep a ball, which falls from above, centered on a tray held by its end-effector (Fig. 2b);
  3. *Ball pushing*: the manipulator is required to push a ball towards a target hole on a table (Fig. 2c);
- *Quadruped robot (Anymal AnyRobotics (2023))*:
  4. *Velocity tracking*: the robot needs to walk at a specified velocity without toppling over (Fig. 2d);
  5. *Running*: the robot should run straight forward as fast as possible without falling;
  6. *Walking to target*: the robot has to walk to a predetermined position;
- *Quadcopter (Crazyflie BitCraze (2023))*:
  7. *Hovering*: the quadcopter should fly to and hover at a designated position (Fig. 2e);
  8. *Flying through a wind field*: the quadcopter needs to reach a target while flying through a wind field;
  9. *Velocity tracking*: the quadcopter should maintain a specified velocity during flight;

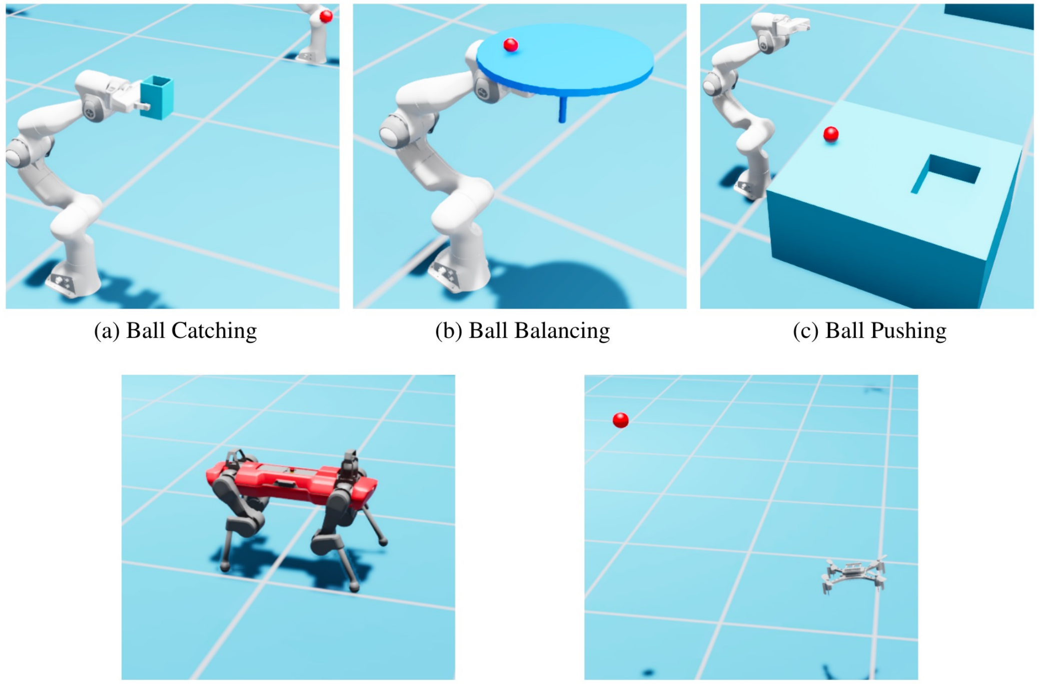
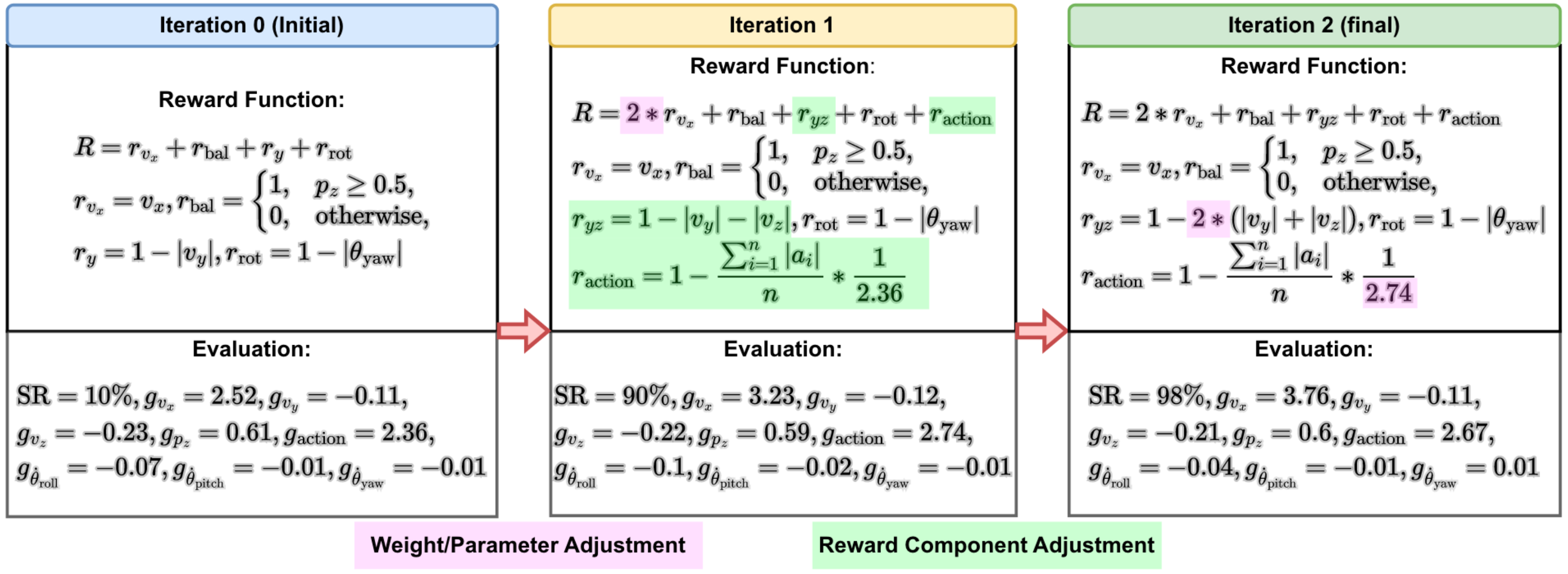


Figure 2: Continuous robotic control tasks with three diverse robotic systems: robotic manipulator (Franka Emika Panda Emika (2023)), quadruped robot (Anymal AnyRobotics (2023)) and quadcopter (Crazyflie BitCraze (2023)). Simulations are conducted in NVIDIA Isaac Sim NVIDIA (2021).

Experimental Results



Iteration 0 (Initial)

Reward Function:

$$R = r_{v_x} + r_{\text{bal}} + r_y + r_{\text{rot}}$$

$$r_{v_x} = v_x, r_{\text{bal}} = \begin{cases} 1, & p_z \geq 0.5, \\ 0, & \text{otherwise}, \end{cases}$$

$$r_y = 1 - |v_y|, r_{\text{rot}} = 1 - |\theta_{\text{yaw}}|$$

Evaluation:

$$\text{SR} = 10\%, g_{v_x} = 2.52, g_{v_y} = -0.11,$$
$$g_{v_z} = -0.23, g_{p_z} = 0.61, g_{\text{action}} = 2.36,$$
$$g_{\dot\theta_{\text{roll}}} = -0.07, g_{\dot\theta_{\text{pitch}}} = -0.01, g_{\dot\theta_{\text{yaw}}} = -0.01$$

Iteration 1

Reward Function:

$$R = 2 * r_{v_x} + r_{\text{bal}} + r_{yz} + r_{\text{rot}} + r_{\text{action}}$$

$$r_{v_x} = v_x, r_{\text{bal}} = \begin{cases} 1, & p_z \geq 0.5, \\ 0, & \text{otherwise}, \end{cases}$$

$$r_{yz} = 1 - |v_y| - |v_z|, r_{\text{rot}} = 1 - |\theta_{\text{yaw}}|$$

$$r_{\text{action}} = 1 - \frac{\sum_{i=1}^{n} |a_i|}{n} * \frac{1}{2.36}$$

Evaluation:

$$\text{SR} = 90\%, g_{v_x} = 3.23, g_{v_y} = -0.12,$$
$$g_{v_z} = -0.22, g_{p_z} = 0.59, g_{\text{action}} = 2.74,$$
$$g_{\dot\theta_{\text{roll}}} = -0.1, g_{\dot\theta_{\text{pitch}}} = -0.02, g_{\dot\theta_{\text{yaw}}} = -0.01$$

Iteration 2 (final)

Reward Function:

$$R = 2 * r_{v_x} + r_{\text{bal}} + r_{yz} + r_{\text{rot}} + r_{\text{action}}$$

$$r_{v_x} = v_x, r_{\text{bal}} = \begin{cases} 1, & p_z \geq 0.5, \\ 0, & \text{otherwise}, \end{cases}$$

$$r_{yz} = 1 - 2 * (|v_y| + |v_z|), r_{\text{rot}} = 1 - |\theta_{\text{yaw}}|$$

$$r_{\text{action}} = 1 - \frac{\sum_{i=1}^{n} |a_i|}{n} * \frac{1}{2.74}$$

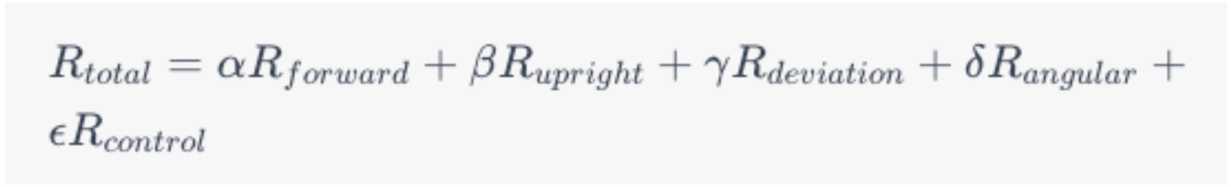
Evaluation:

$$\text{SR} = 98\%, g_{v_x} = 3.76, g_{v_y} = -0.11,$$
$$g_{v_z} = -0.21, g_{p_z} = 0.6, g_{\text{action}} = 2.67,$$
$$g_{\dot\theta_{\text{roll}}} = -0.04, g_{\dot\theta_{\text{pitch}}} = -0.01, g_{\dot\theta_{\text{yaw}}} = 0.01$$

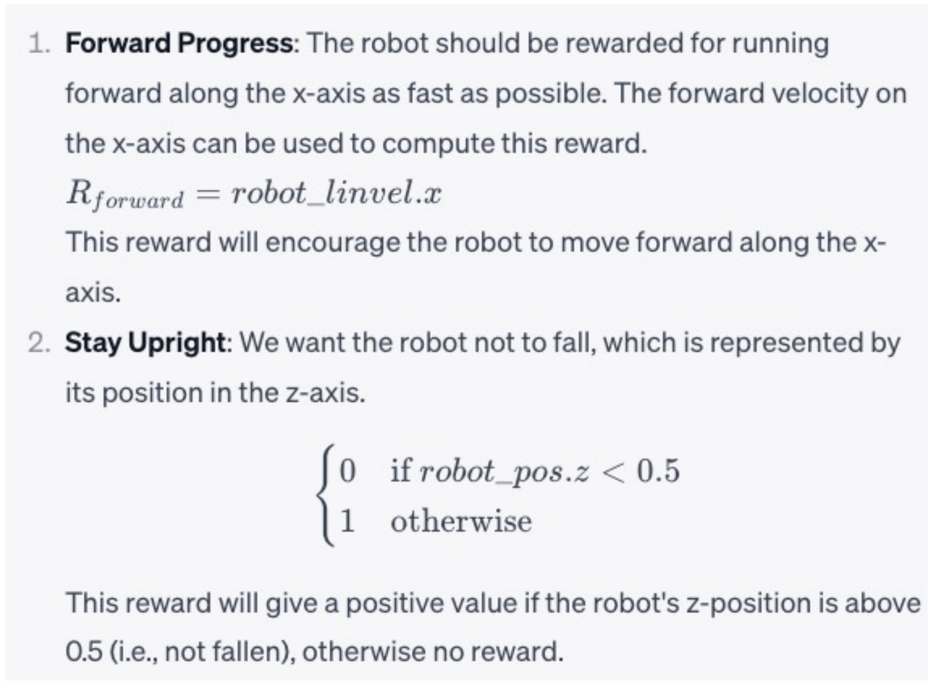
Figure 3: Reward functions in different self-refinement iterations for the quadruped robot forward running task.

Experimental Results – Reward by ChatGPT

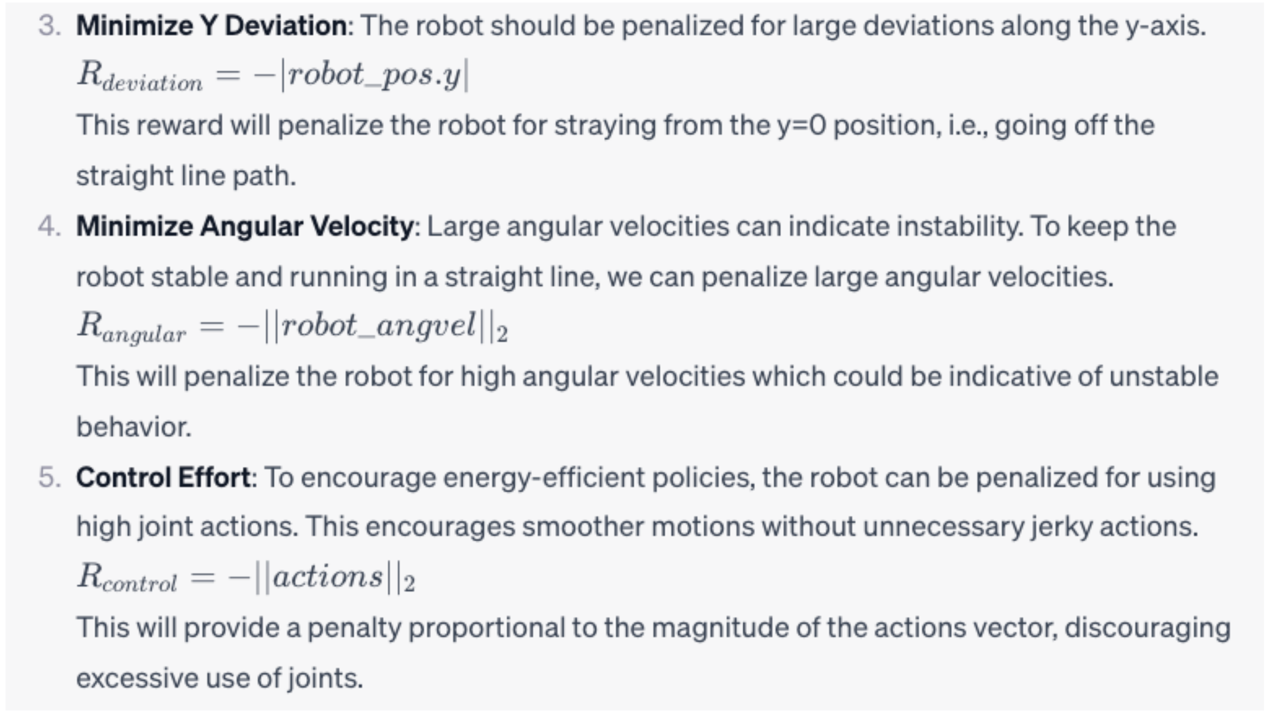
Yang feeds the prompt in the Appendix to ChatGPT (the default GPT model)



$$R_{total} = \alpha R_{forward} + \beta R_{upright} + \gamma R_{deviation} + \delta R_{angular} + \epsilon R_{control}$$



1. **Forward Progress**: The robot should be rewarded for running forward along the x-axis as fast as possible. The forward velocity on the x-axis can be used to compute this reward.
   $R_{forward} = robot\_linvel.x$
   This reward will encourage the robot to move forward along the x-axis.
2. **Stay Upright**: We want the robot not to fall, which is represented by its position in the z-axis.
   $$\begin{cases} 0 & \text{if } robot\_pos.z < 0.5 \\ 1 & \text{otherwise} \end{cases}$$
   This reward will give a positive value if the robot's z-position is above 0.5 (i.e., not fallen), otherwise no reward.



3. **Minimize Y Deviation**: The robot should be penalized for large deviations along the y-axis.
   $R_{deviation} = -|robot\_pos.y|$
   This reward will penalize the robot for straying from the y=0 position, i.e., going off the straight line path.
4. **Minimize Angular Velocity**: Large angular velocities can indicate instability. To keep the robot stable and running in a straight line, we can penalize large angular velocities.
   $R_{angular} = -||robot\_angvel||_2$
   This will penalize the robot for high angular velocities which could be indicative of unstable behavior.
5. **Control Effort**: To encourage energy-efficient policies, the robot can be penalized for using high joint actions. This encourages smoother motions without unnecessary jerky actions.
   $R_{control} = -||actions||_2$
   This will provide a penalty proportional to the magnitude of the actions vector, discouraging excessive use of joints.

The University of Manchester

Experimental Results

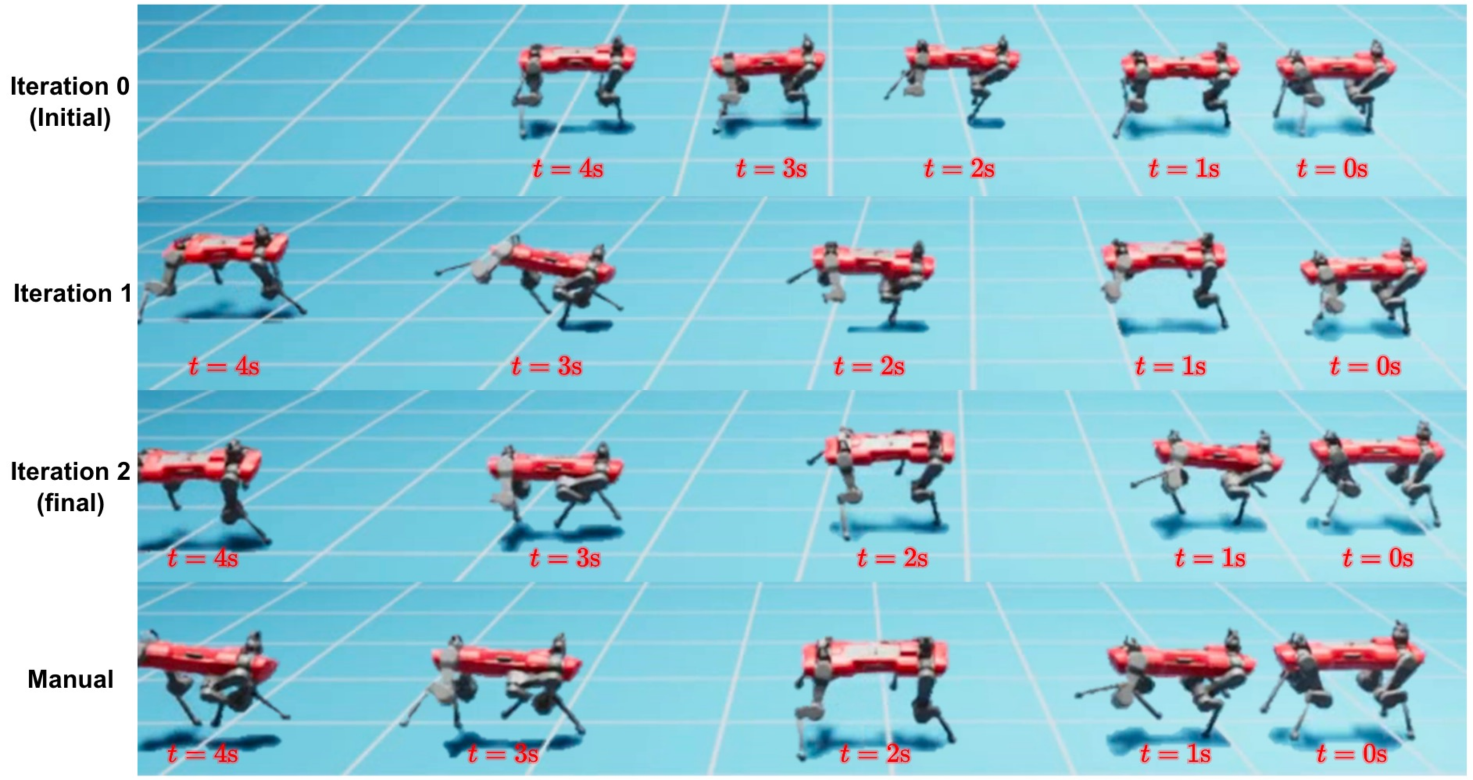


Figure 4: System behaviors corresponding to reward functions in different self-refinement iterations, as well as the manually designed reward function. The time interval between each displayed point is set to 1s.

MANCHESTER 1824
The University of Manchester

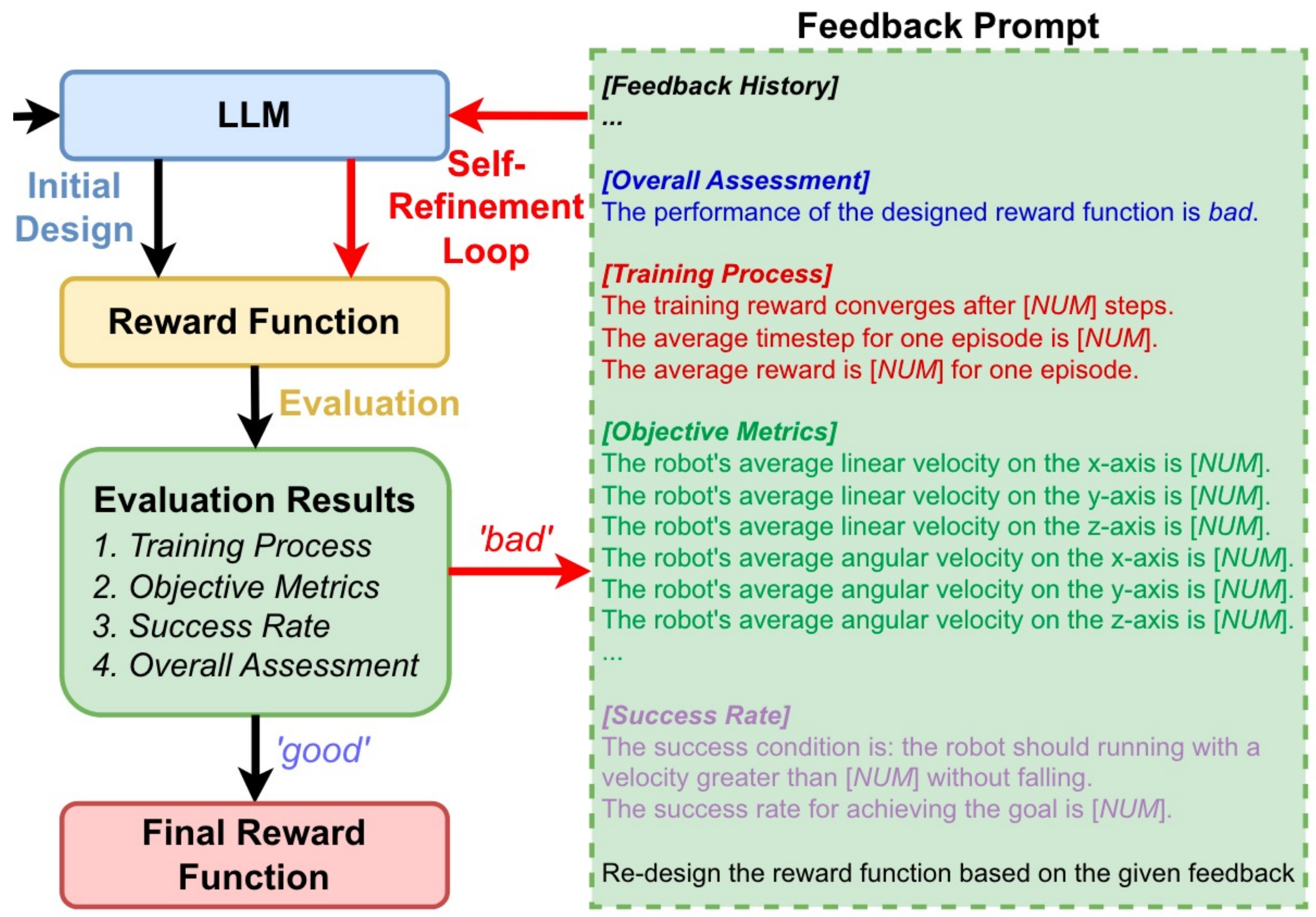
Experimental Results

Table 1: Success rates of different reward functions and the number of self-refinement iterations (Iter.) used for R_{Refined}.

Robotic System	Task	Success Rate SR			Iter.
		R_{Initial}	R_{Refined}	R_{Manual}	
Manipulator	Ball Catching	100%	100%	100%	0
	Ball Balancing	100%	100%	98%	0
	Ball Pushing	0%	93%	95%	5
Quadruped	Velocity Tracking	0%	96%	92%	3
	Running	10%	98%	95%	2
	Walking to Target	0%	85%	80%	5
Quadcopter	Hovering	0%	98%	92%	2
	Wind Field	0%	100%	100%	4
	Velocity Tracking	0%	99%	91%	3

The University of Manchester

Conclusion



Feedback Prompt

[Feedback History]
...

[Overall Assessment]
The performance of the designed reward function is *bad*.

[Training Process]
The training reward converges after [*NUM*] steps.
The average timestep for one episode is [*NUM*].
The average reward is [*NUM*] for one episode.

[Objective Metrics]
The robot's average linear velocity on the x-axis is [*NUM*].
The robot's average linear velocity on the y-axis is [*NUM*].
The robot's average linear velocity on the z-axis is [*NUM*].
The robot's average angular velocity on the x-axis is [*NUM*].
The robot's average angular velocity on the y-axis is [*NUM*].
The robot's average angular velocity on the z-axis is [*NUM*].
...

[Success Rate]
The success condition is: the robot should running with a velocity greater than [*NUM*] without falling.
The success rate for achieving the goal is [*NUM*].

Re-design the reward function based on the given feedback

A self-refined LLM framework as an **automated reward function** designer for DRL in continuous robotic control tasks.

Evaluate the proposed framework across **nine diverse robotic control tasks**, distributed among **three distinct robotic systems.**

Limitations:

1. **Coarse Reward Design**: inability to address nuanced aspects of desired system behaviours that are difficult to quantify through the automated evaluation process, such as the gait of a quadruped robot
2. **Rely on pre-trained common-sense knowledge**: For tasks that are highly specialized or not represented in its training data, the LLM may struggle to devise an appropriate reward function