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

Jiayang Song[1,†], Zhehua Zhou[1,†], Jiawei Liu[2], Chunrong Fang[2], Zhan Shu[1], and Lei Ma[3,1]

***Abstract*— Although Deep Reinforcement Learning (DRL) has achieved notable success in numerous robotic applications, designing a high-performing reward function remains a challenging task that often requires substantial manual input. Recently, Large Language Models (LLMs) have been extensively adopted to address tasks demanding in-depth common-sense knowledge, such as reasoning and planning. Recognizing that reward function design is also inherently linked to such knowledge, LLM offers a promising potential in this context. Motivated by this, we propose in this work a novel LLM framework with a self-refinement mechanism for automated reward function design. The framework commences with the LLM formulating an initial reward function based on natural language inputs. Then, the performance of the reward function is assessed, and the results are presented back to the LLM for guiding its self-refinement process. We examine the performance of our proposed framework through a variety of continuous robotic control tasks across three diverse robotic systems. The results indicate that our LLM-designed reward functions are able to rival or even surpass manually designed reward functions, highlighting the efficacy and applicability of our approach.**

I. Introduction

Over the past years, substantial progress has been achieved in leveraging Deep Reinforcement Learning (DRL) to tackle a broad spectrum of complex challenges across diverse robotic domains, such as manipulation [1], navigation [2], locomotion [3], and aerial robotics [4]. However, despite these advancements, training high-performing DRL agents remains a challenging task [5]. A principal contributing factor to this complexity is the inherent difficulty in designing an effective reward function, which is vital and fundamental to DRL approaches [6].

Conventional methods of reward function design predominantly rely on meticulous manual crafting [7]. Recent research has introduced Automated Reinforcement Learning (AutoRL) approaches [8], aiming to automate the hyper-parameters and reward function tuning in DRL. These approaches commence with a predefined, parameterized reward function and subsequently fine-tune its parameters to identify an optimal reward function [9], [10]. However, instead of developing the reward function from scratch, AutoRL remains dependent on an initial parameterized function provided by human experts. The construction of such a function often demands domain-specific expertise and a significant investment of time and effort.

In recent research, Large Language Models (LLMs) have been increasingly utilized for tasks that demand common-sense reasoning and extensive world knowledge [11], spanning domains like natural language processing [12], task planning [13], and reasoning [14]. The compelling outcomes from these studies reveal the ability of LLMs to emulate human cognitive processes and integrate a substantial degree of common-sense knowledge [15], [16]. Given that designing reward functions often also depends on such knowledge, researchers are currently exploring the potential of LLMs as reward function designers for DRL. Leveraging natural language instructions as input, LLMs are able to formulate effective reward functions for simple tasks in game environments with discrete action spaces [17]. Moreover, their rich internalized knowledge about the world also aids in comprehending both user preferences and task requirements. However, as LLMs are essentially engineered to generate word sequences that align with human-like context, their efficacy and reliability in reward function design remain uncertain, especially for robotic control tasks that involve continuous action spaces.

In this work, we investigate the possibility of employing LLM as an automated reward function designer for DRL-driven continuous robotic control tasks. Motivated by recent studies that demonstrate the capability of LLM for self-refinement [18], [19], we propose a novel self-refined LLM framework for reward function design. The framework consists of three steps (see Fig. 1): 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. We examine the performance of the proposed self-refined LLM framework across nine different tasks distributed among three diverse robotic systems. The results show that our approach is capable of generating reward functions that not only induce desired robotic behaviors but also rival or even exceed those meticulously hand-crafted reward functions.

The contributions of this paper are threefold:

- 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

†Contribute equally to this paper.

[1]JS, ZZ and ZS are with University of Alberta, Canada. Emails: {jiayan13, zhehua1, zshu1}@ualberta.ca

[2]JL and CF are with Nanjing University, China. Emails: jw.liu@smail.nju.edu.cn, fangchunrong@nju.edu.cn

[3]LM is with The University of Tokyo, Japan, and University of Alberta, Canada. Email: ma.lei@acm.org

All codes and results relevant to this paper are available at `https://github.com/zhehuazhou/LLM_Reward_Design`.

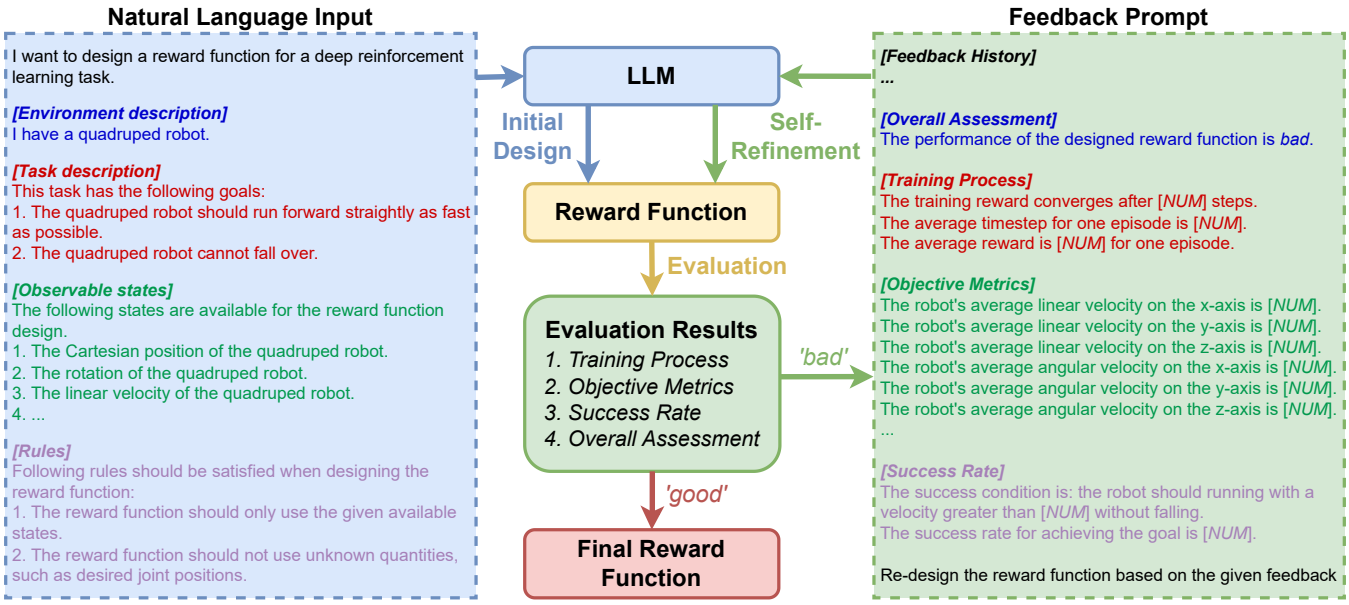


Fig. 1: Our proposed self-refine LLM framework for reward function design. It consists of three steps: *initial design*, *evaluation*, and *self-refinement loop*. A quadruped robot forward running task is used as an example here. A complete list of the prompts used in this work can be found in the preprint version of this paper [20].

function designer.

- We incorporate a self-refinement mechanism into the reward function design process to enhance its outcomes.
- We highlight the effectiveness and applicability of our proposed approach through a variety of continuous robotic control tasks across diverse robotic systems.

II. Related Work

Reward Function Design with AutoRL AutoRL extends Automated Machine Learning (AutoML) [21] principles to address reinforcement learning challenges. Its primary focus is to automate the fine-tuning of both the architectures of neural networks and the hyperparameters of learning algorithms [8]. For reward function design, AutoRL utilizes evolutionary algorithms [22] to adjust the parameters of a predefined parameterized reward function, aiming to identify an optimal reward function. In [9], AutoRL is applied to modify the reward function for a navigation problem, while [10] further expands the use of AutoRL in reward function optimization to multiple reinforcement learning benchmarks simulated in Mujoco [23]. In essence, AutoRL can be considered a reward shaping technique [24]. However, due to its dependency on an initially hand-crafted parameterized reward function, AutoRL lacks the ability to formulate a reward function entirely from scratch.

Reward Function Design with LLM Benefiting from its pre-trained common-sense knowledge, LLM offers the potential to alleviate the human effort required in formulating reward functions. Recent studies have revealed the capability of LLM in directing reward shaping approaches [25], [26]. In [27], instead of creating a reward function for DRL, LLM is employed to determine objective functions for predefined model predictive controllers. State-of-the-art research demonstrates that for simpler tasks, such as normal-form games [28] with discrete action spaces, LLM can serve directly as a proxy reward function [17]. Through processing natural language instructions, LLM seamlessly integrates task requirements and user preferences into reward functions [29]. However, whether LLM is able to independently design a reward function from scratch for continuous robotic control tasks remains an open research question.

III. Preliminary

DRL Setup We model the DRL problem as a Partially Observable Markov Decision Process (POMDP) [30] $\mathcal{M}(S, O, A, T, R, \gamma)$ with continuous state and action spaces. Given a state $s \in S$, a DRL agent determines an action $a \in A$. The system then transitions to a new state s' according to the transition distribution function $T(s'|s, a)$, which results in an observation $o \in O$. For a given reward function $R : S \times A \to \mathbb{R}$, the training process of DRL aims to find a policy $\hat{\pi}_R$ that maximizes the expected cumulative discounted reward $\hat{\pi}_R = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where γ is the discount factor.

Reward Function Design While the reward function R provides immediate feedback to the DRL agent, it often lacks human interpretability due to the inherent difficulty in directly associating numerical values with system behaviors. To analyze the performance of a policy, humans typically evaluate system trajectories $\mathcal{T}$ generated by the trained DRL agent through a performance metric $G(\mathcal{T})$. For example, in the humanoid walking task, the performance metric $G(\mathcal{T})$ could be the maximum distance the humanoid can travel without falling. Therefore, the goal in designing a reward function is to determine an optimal reward function $\hat{R}$ that, after the training process, results in a policy $\hat{\pi}_{\hat{R}}$ that

maximizes the performance metric $G(\mathcal{T})$.

IV. Self-Refined LLM for Reward Function Design

In this section, we introduce a self-refined LLM framework for automated reward function design. It contains three steps: *initial design*, *evaluation*, and *self-refinement loop*.

A. Initial Design

We first employ the LLM to formulate an initial reward function based on natural language input. To enhance the LLM's comprehension of the robotic control task, we segment the natural language prompt into four parts (see Fig. 1):

- *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.

Similar to many hand-crafted reward functions, the initial reward function formulated by the LLM is often given as a weighted combination of multiple individual reward components, i.e., we have $R = \sum_{i=0}^{n} w_i r_i$. However, the initial weights w_i are usually unreliable and necessitate adjustments. We address this challenge by using our proposed self-refinement process.

It is worth mentioning that while many studies leverage few-shot in-context learning [12] to guide the LLM in generating responses in a desired manner, our approach utilizes the LLM as a zero-shot reward function designer, excluding examples in our prompts. The major reason is that, due to the inherent task-specificity of reward function design, finding universally applicable examples for a diverse array of robotic control tasks proves challenging. To ensure the performance of the designed reward function, we employ the subsequent evaluation and self-refinement processes. A complete list of the prompts used in our experiments is available in [20].

B. Evaluation

After the LLM determines the reward function R, we assess its efficacy via an evaluation process (see Fig. 1). Aiming to minimize human intervention, the evaluation is structured as an automated procedure.

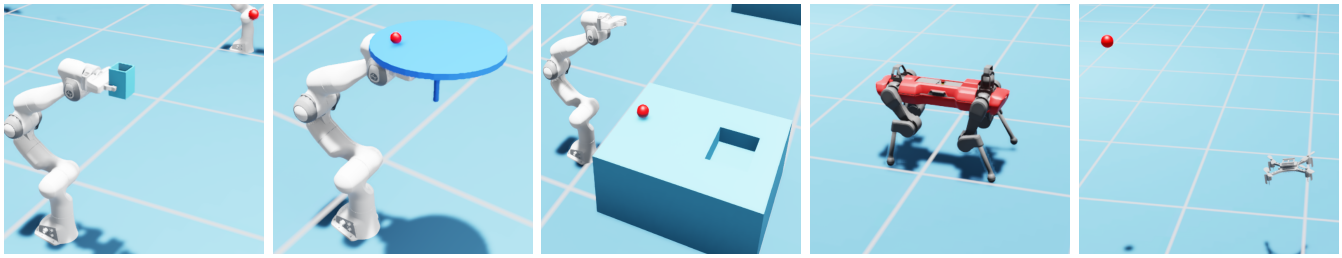
We begin by initiating a training process to obtain a trained optimal DRL policy $\hat{\pi}_R$. Subsequently, we sample n_t trajectories $\mathcal{T}_i, i = 1, \ldots, n_t$ of this trained policy $\hat{\pi}_R$, each originating from a distinct, randomly selected initial state. Performance of the reward function R is then evaluated from the following three aspects:

- *Training process:* we first summarize the training process for the policy $\hat{\pi}_R$ to evaluate the immediate effectiveness of the designed reward function R. 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 $G(\mathcal{T})$ with multiple individual task-specific objective metrics $g_k(\mathcal{T}), k = 1, \ldots, n_g$. Each objective metric $g_k(\mathcal{T})$ addresses an aspect of the task requirements. For instance, in the quadruped robot's straightforward 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 $g_k(\mathcal{T})$ 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 $\hat{\pi}_R$ 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) [35] 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.

As a conclusion of the evaluation, we finally categorize the overall performance of the designed reward function R as either '*good*' or '*bad*'. Given that the training process and objective metrics are intrinsically task-dependent, it is challenging to establish a universally applicable standard to assess different tasks based on these two criteria. Therefore, we rely solely on the success rate SR for the overall assessment and reserve other details as guidance for the subsequent self-refinement process. If the success rate SR exceeds a predefined threshold, the performance of the reward function is considered '*good*'. Otherwise, we label it as '*bad*' and initiate a self-refinement process to improve.

C. Self-Refinement Loop

To enhance the designed reward function, we employ a self-refinement process. It starts with the construction of a feedback prompt for the LLM based on evaluation results (see Fig. 1). To offer the LLM a clear and immediate comprehension of the performance of the reward function, we position the overall assessment at the beginning of the prompt, followed by detailed information of the training process, objective metrics, and success rate. Guided by this feedback and previous feedback history, the LLM attempts to develop an updated reward function. Details about all feedback prompts used in our experiments are presented in [20].



(a) Ball Catching (b) Ball Balancing (c) Ball Pushing (d) Velocity Tracking (e) Hovering

Fig. 2: Continuous robotic control tasks with three diverse robotic systems: robotic manipulator (Franka Emika Panda [31]), quadruped robot (Anymal [32]) and quadcopter (Crazyflie [33]). Simulations are conducted in NVIDIA Isaac Sim [34].

For finding an optimal reward function, we repeat the evaluation and self-refinement processes in a loop until either a predefined maximum number of iterations is reached, or the evaluation suggests '*good*' performance. The reward function, resulting from the self-refinement loop, is accepted as the final designed reward function.

V. Experimental Results

A. Experimental Setup

We evaluate the performance of our proposed framework in designing reward functions through nine distinct continuous robotic control tasks across three diverse robotic systems (see Fig. 2). Specifically, we employ the following tasks and systems that are also frequently referenced as benchmark challenges in DRL studies [36], [37], [38]:

- *Robotic manipulator (Franka Emika Panda [31])*:
  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 [32])*:
  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 [33])*:
  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;

For each task, we compare the reward functions obtained by using three different methods: 1) R_{Initial}, which is the LLM's initial design of the reward function based on natural language input; 2) R_{Refined}, which is the final reward function formulated by the proposed self-refined LLM framework; 3) R_{Manual}, which is a manually designed reward function sourced from existing literature or benchmarks.

In the evaluation process, we utilize the Proximal Policy Optimization (PPO) [39] as the DRL algorithm to find the optimal policy $\hat{\pi}_R$ for each reward function. To concentrate on analyzing the reward function, we use the same learning parameters and neural network architectures in the training processes for all three reward functions. These parameters are derived by fine-tuning based on the manually designed reward function to achieve its optimal performance. For each trained policy $\hat{\pi}_R$, we sample $n_t = 100$ trajectories and compute the corresponding objective metrics and success rates. The success rate threshold for the overall assessment is set at 95%, and the maximum number of self-refinement iterations is selected as 5.

We simulate the robotic control tasks with NVIDIA Isaac Sim [34], [40] and employ GPT4 as the underlying LLM. All experiments are conducted on a laptop equipped with an Intel® Core™ i7-10870H CPU and an NVIDIA RTX 3080 Max-Q GPU with 16 GB VRAM. Further details regarding the experimental setup are given in [20].

B. Reward Function and Objective Metrics

We use the quadruped robot forward running task as an example to illustrate the reward function design process via our proposed self-refined LLM framework. The observable states for this task are: the global positions of the robot's base p_x, p_y, p_z; the linear velocities of the robot v_x, v_y, v_z; the base rotations relative to the world frame $\theta_{\text{roll}}, \theta_{\text{pitch}}, \theta_{\text{yaw}}$, the angular velocities $\dot{\theta}_{\text{roll}}, \dot{\theta}_{\text{pitch}}, \dot{\theta}_{\text{yaw}}$, and the current action command for the 12 joints $a_i, i = 1\ldots,12$. The LLM needs to determine which of these observable states should be incorporated into the reward function. The STL expression representing the success condition is given as $\varphi \equiv \square_{[0.8,5]}(v_x \geq 2) \wedge \square_{[0,5]}((p_y \leq 2) \wedge (p_z \geq 0.5))$, which indicates that following an initial acceleration phase lasting 0.8 seconds, the robot must always maintain a speed of at least $v_x = 2$ m/s until the simulation stops at $t = 5$ seconds. Meanwhile, the robot must restrict lateral deviation to under 2 m and cannot fall over. The objective metrics used in the evaluation process are: the average linear velocities $g_{v_x}, g_{v_y}, g_{v_z}$; the average z-position g_{p_z}; the average normalized action values g_{action}; and the average angular velocities $g_{\dot{\theta}_{\text{roll}}}, g_{\dot{\theta}_{\text{pitch}}}, g_{\dot{\theta}_{\text{yaw}}}$. For comparison, we employ a

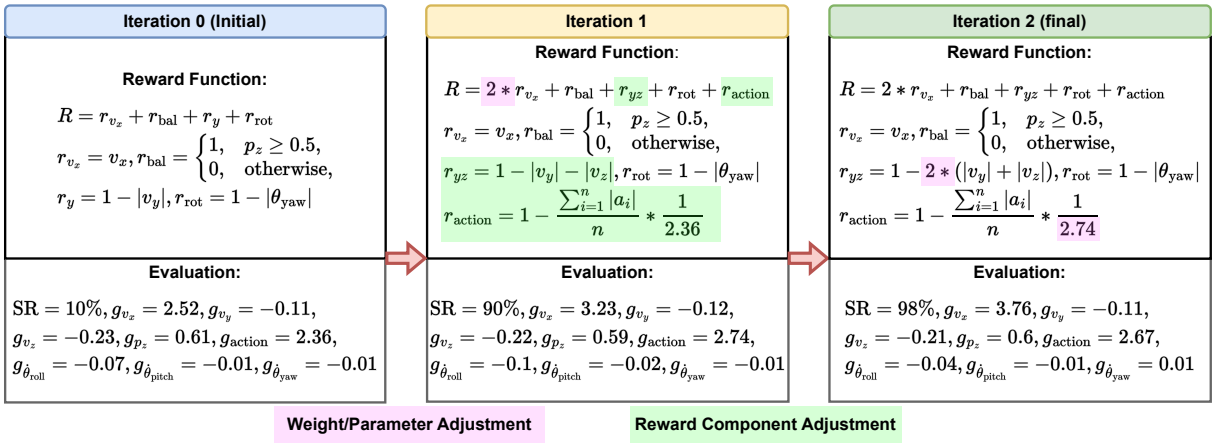


Fig. 3: Reward functions in different self-refinement iterations for the quadruped robot forward running task.

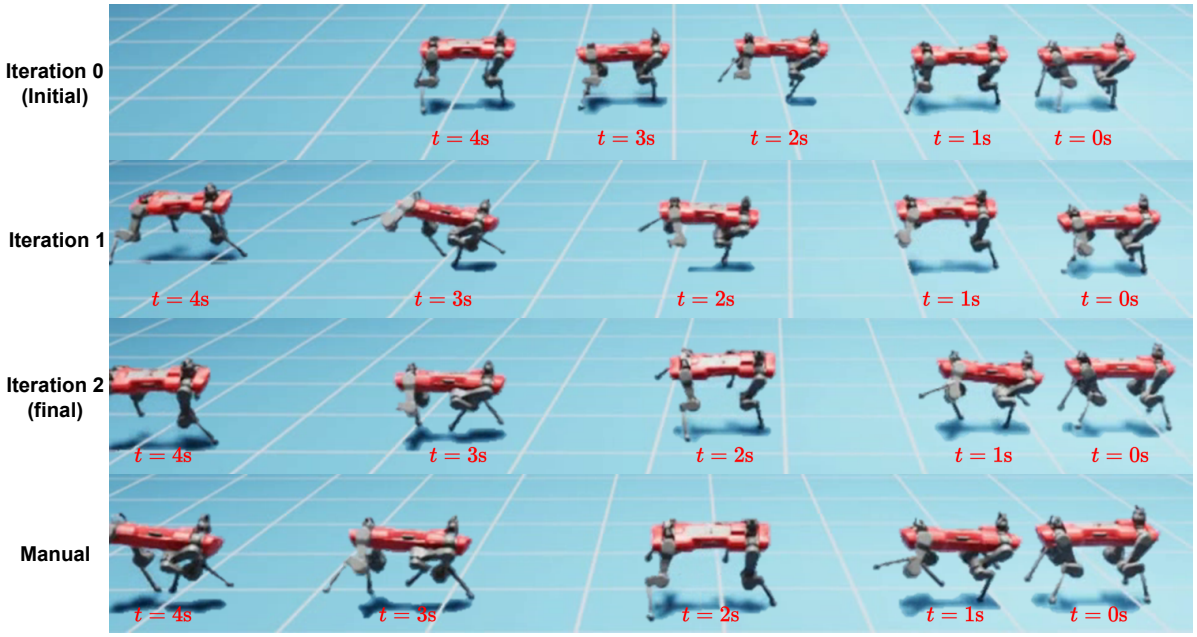


Fig. 4: System behaviors corresponding to reward functions in different self-refinement iterations, as well as the manually designed reward function. The time interval between each displayed point is set to 1s.

manually designed reward function based on [40], which is given as $R = 1.5v_x + 0.2r_{\text{bal}} - 0.5\frac{|p_y|}{2} - 0.1|\dot{\theta}_{\text{yaw}}|$, with $r_{\text{bal}} = 1$ if $p_z \geq 0.5$ and $r_{\text{bal}} = 0$ otherwise.

The LLM requires a total of two self-refinement iterations to identify a satisfactory reward function. Fig. 3 presents the reward functions in different iterations alongside their respective evaluation outcomes. Corresponding system behaviors are given in Fig. 4, which also illustrates the behavior produced by the manually designed reward function.

Similar to a manual refinement process, the LLM adjusts the reward function by altering its weights or parameters or by modifying the structure of its components. Initially, the reward function contains four components, emphasizing forward velocity and balance. However, this initial design proves insufficient, as the robot overly prioritizes maintaining balance over running forward, leading to a low forward velocity and success rate (see Iteration 0 in Fig. 4). Responding to this feedback, our proposed framework then initiates a self-refinement iteration. It increases the weight attributed to forward velocity v_x and adjusts the penalty associated with lateral deviation. Furthermore, it introduces a penalty for large actions in reaction to the action metric g_{action} contained in the feedback. This refinement enhances performance, elevating the success rate to 90%. However, the evaluation indicates that the robot still takes aggressive actions to achieve high velocity (e.g., $t = 3\text{s}$ of Iteration 1 in Fig. 4). Hence, in its second self-refinement iteration, the LLM increases the penalties for excessive actions and lateral deviations. The resulting reward function leads to a behavior that closely aligns with the behavior obtained from a manually designed reward function (see Iteration 2 and Manual in Fig. 4). The manually designed reward function yields the following evaluation metrics: $\text{SR} = 95\%, g_{v_x} = 3.748, g_{v_y} = -0.105, g_{v_z} = -0.223, g_{p_z} = 0.609, g_{\text{action}} = 2.673, g_{\dot{\theta}_{\text{roll}}} = -0.071, g_{\dot{\theta}_{\text{pitch}}} = -0.019, g_{\dot{\theta}_{\text{yaw}}} = 0.041$, which are also similar to those of the final reward function

TABLE I: 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

formulated by the LLM. This justifies the efficacy of the proposed self-refined LLM framework in designing reward functions for continuous robotic control tasks.

Other control tasks exhibit a similar pattern as the quadruped robot forward running task. See [20] and the supplementary video for details on these tasks.

C. Success Rates

We further evaluate the success rates across all the tasks under consideration to assess the generalizability and applicability of our proposed approach. The results are summarized in Table I. Detailed information regarding the employed success conditions for each task is presented in [20].

It can be observed that the initial reward function demonstrates a binary level of performance. For tasks with straightforward objectives, such as ball catching or balancing, the LLM is able to devise a high-performing reward function on its first attempt. Conversely, for more complex tasks that involve multiple objectives, e.g., ensuring a quadruped robot maintains a set velocity while walking straight and keeping balance, the initial reward function often registers a success rate of 0%. In such cases, the LLM predominantly relies on feedback to understand the implications of its design, necessitating multiple self-refinement iterations. By leveraging the evaluation results, the LLM is capable of effectively revising its reward function design. As a result, it achieves success rates that match or even surpass those of manually designed reward functions for all examined tasks.

However, the performance is affected by the intrinsic complexities of the task. For tasks demanding intricate reward function components, e.g., the quadruped robot walking to target task, the success rate diminishes, indicating a need for further self-refinement iterations or even detailed human feedback. Nevertheless, even in these challenging scenarios, our self-refined LLM framework consistently identifies reward functions that outperform manual designs. This illustrates the broad applicability of our approach across a wide range of continuous robotic control tasks.

VI. Discussion

Learning Parameters and AutoRL In our experiments, we observe that during the self-refinement process, the LLM often has to adjust the weights of reward components. While the LLM is able to determine a final reward function that yields desired system behavior, the weights it assigns are not guaranteed to be optimal. In other words, there might exist configurations that produce even better outcomes. One potential improvement would be to integrate the LLM with AutoRL. Once the LLM formulates the reward function, AutoRL could optimize its parameters using search-based approaches. In such a case, the LLM serves as an initial designer, offering a parameterized reward function to AutoRL. This strategy can further be extended to fine-tune learning parameters and neural network architectures. By identifying optimal parameters before each self-refinement iteration, the LLM can then focus on adjusting the structural components of the reward function. However, adopting this approach could greatly prolong the reward function design process.

Fine-tuned LLM Recent studies indicate that fine-tuning the LLM for specific tasks can greatly enhance its performance [41], [42], [43]. Such a technique could also improve our approach. The complexity associated with comprehending control task requirements and formulating appropriate reward functions could potentially be alleviated by deploying an LLM specifically fine-tuned for reward function design, as opposed to a general-purpose model. Nonetheless, fine-tuning an LLM typically demands substantial resources, and garnering enough training data for reward function design might also be challenging.

Limitations One major limitation of our approach is its inability to address nuanced aspects of desired system behaviors that are difficult to quantify through the automated evaluation process, such as the gait of a quadruped robot. Addressing this challenge often necessitates human intervention. By offering detailed human feedback, the LLM is capable of fine-tuning its outcome accordingly, as illustrated in [27]. Another limitation is the reliance of the LLM on its 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. Under such circumstances, enhancing the natural language input prompt with more details about the specific robotic system and control task becomes essential.

VII. Conclusion

In this paper, we introduce a self-refined LLM framework as an automated reward function designer for DRL in continuous robotic control tasks. The framework operates in three steps: First, the LLM devises an initial reward function by using a natural language input. Second, an automated evaluation process is initiated to assess the performance of the designed reward function. Third, based on the evaluation results, a feedback prompt is provided to the LLM, guiding its self-refinement process of the reward function. We evaluate our proposed framework across nine diverse robotic control tasks, distributed among three distinct robotic systems. The results indicate that our approach is able to generate reward functions that are on par with, or even superior to, those manually designed ones. For future work, we plan to integrate the LLM with AutoRL techniques, enabling not only the reward function, but also all learning parameters to be designed autonomously.

REFERENCES

[1] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation," in *IEEE International Conference on Robotic Computing*. IEEE, 2019, pp. 590–595.

[2] K. Zhu and T. Zhang, "Deep reinforcement learning based mobile robot navigation: A review," *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

[3] J. Yue, "Learning locomotion for legged robots based on reinforcement learning: A survey," in *International Conference on Electrical Engineering and Control Technologies*. IEEE, 2020, pp. 1–7.

[4] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed *et al.*, "Drone deep reinforcement learning: A review," *Electronics*, vol. 10, no. 9, p. 999, 2021.

[5] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski *et al.*, "What matters for on-policy deep actor-critic methods? a large-scale study," in *International Conference on Learning Representations*, 2020.

[6] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

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

[8] J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust *et al.*, "Automated reinforcement learning (autorl): A survey and open problems," *Journal of Artificial Intelligence Research*, vol. 74, pp. 517–568, 2022.

[9] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[10] A. Faust, A. Francis, and D. Mehta, "Evolving rewards to automate reinforcement learning," *arXiv preprint arXiv:1905.07628*, 2019.

[11] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[13] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[14] E. Zelikman, Y. Wu, J. Mu, and N. Goodman, "Star: Bootstrapping reasoning with reasoning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 476–15 488, 2022.

[15] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?" *arXiv preprint arXiv:1909.01066*, 2019.

[16] J. Davison, J. Feldman, and A. M. Rush, "Commonsense knowledge mining from pretrained models," in *Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019, pp. 1173–1178.

[17] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," *arXiv preprint arXiv:2303.00001*, 2023.

[18] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang *et al.*, "Self-refine: Iterative refinement with self-feedback," *arXiv preprint arXiv:2303.17651*, 2023.

[19] J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han, "Large language models can self-improve," *arXiv preprint arXiv:2210.11610*, 2022.

[20] J. Song, Z. Zhou, J. Liu, Z. Shu, and L. Ma, "Self-refined large language model as automated reward function designer for deep reinforcement learning in robotics," *arXiv preprint*, 2023.

[21] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[23] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[24] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *International Conference on Machine Learning*, vol. 99. Citeseer, 1999, pp. 278–287.

[25] S. Mirchandani, S. Karamcheti, and D. Sadigh, "Ella: Exploration through learned language abstraction," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 529–29 540, 2021.

[26] T. Carta, P.-Y. Oudeyer, O. Sigaud, and S. Lamprier, "Eager: Asking and answering questions for automatic reward shaping in language-guided rl," *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 478–12 490, 2022.

[27] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik *et al.*, "Language to rewards for robotic skill synthesis," *arXiv preprint arXiv:2306.08647*, 2023.

[28] M. Costa-Gomes, V. P. Crawford, and B. Broseta, "Cognition and behavior in normal-form games: An experimental study," *Econometrica*, vol. 69, no. 5, pp. 1193–1235, 2001.

[29] H. Hu and D. Sadigh, "Language instructed reinforcement learning for human-ai coordination," *arXiv preprint arXiv:2304.07297*, 2023.

[30] G. E. Monahan, "State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.

[31] F. Emika, "Franka emika," 2023. [Online]. Available: https://www.franka.de/

[32] AnyRobotics, "Anymal," 2023. [Online]. Available: https://www.anybotics.com/robotics/anymal/

[33] BitCraze, "Crazyflie," 2023. [Online]. Available: https://www.bitcraze.io/products/crazyflie-2-1/

[34] NVIDIA, "Nvidia isaac sim," 2021. [Online]. Available: https://developer.nvidia.com/isaac-sim

[35] A. Donzé, "On signal temporal logic," in *International Conference on Runtime Verification*. Springer, 2013, pp. 382–383.

[36] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3019–3026, 2020.

[37] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," *arXiv preprint arXiv:2009.12293*, 2020.

[38] Z. Zhou, J. Song, X. Xie, Z. Shu, L. Ma, D. Liu, J. Yin, and S. See, "Towards building ai-cps with nvidia isaac sim: An industrial benchmark and case study for robotics manipulation," *arXiv preprint arXiv:2308.00055*, 2023.

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

[40] NVIDIA, "Omniverse isaac gym reinforcement learning environment," 2023. [Online]. Available: https://github.com/NVIDIA-Omniverse/OmniIsaacGymEnvs

[41] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2790–2799.

[42] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.

[43] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.