AnyBipe: An Automated End-to-End Framework for Training and Deploying Bipedal Robots Powered by Large Language Models

Yifei Yao[1], Wentao He[2†], Chenyu Gu[1†], Jiaheng Du[1†], Fuwei Tan[1], Zhen Zhu[1], and Jun-Guo Lu[1,*]

***Abstract*— Training and deploying reinforcement learning (RL) policies for robots is a complex task, requiring careful design of reward functions, sim-to-real transfer, and performance evaluation across various robot configurations. These tasks traditionally demand significant human expertise and effort. To address these challenges, this paper introduces *Anybipe*, a novel, fully automated, end-to-end framework for training and deploying bipedal robots, leveraging large language models (LLMs) for reward function generation, while supervising model training, evaluation, and deployment. The framework integrates comprehensive quantitative metrics to assess policy performance, deployment effectiveness, and safety. Additionally, it allows users to incorporate prior knowledge and preferences, improving the accuracy and alignment of generated policies with expectations. We demonstrate how *Anybipe* reduces human labor while maintaining high levels of accuracy and safety, examined on three different bipedal robots, showcasing its potential for autonomous RL training and deployment.**

I. Introduction

With the integration of advanced control algorithms, enhanced physical simulations, and improved computational power, robotics has achieved remarkable progress [1]. These advancements allow robots to tackle tasks from industrial automation to personal assistance with exceptional efficiency and autonomy [2]. As industrial robotics continues to advance, attention has increasingly shifted toward humanoid robots, with researchers prioritizing the replication of human-like traits and the ability to perform tasks designed for humans [3]. In this context, bipedal robots offer a practical means to simulate the structure of human lower limbs, providing a valuable approach to studying and improving locomotion skills in humanoid robotics.

Control strategies for bipedal robots typically draw on either traditional control techniques or reinforcement learning (RL) approaches [4]. Traditional methods depend on abstracting problems, building models, and detailed planning, whereas RL uses reward functions to iteratively steer robots toward completing tasks. By interacting repeatedly with their environments, RL empowers robots to fine-tune their control strategies and develop critical skills. This method proves especially effective in simulated settings, where robots can learn through trial and error to navigate complex terrains and handle disturbances.

Despite these strides, training and deploying reinforcement learning (RL) algorithms remain formidable challenges. Crafting effective reward functions demands careful attention to task-specific objectives and the inclusion of safety constraints for real-world application [5]. This intricacy requires significant engineering effort across training, testing, and refinement stages. While techniques like reward shaping and safe RL hold promise [6], their reliance on prior experience often complicates the design process. Furthermore, the persistent "Sim-to-Real" challenge—stemming from hard-to-measure limitations in physics simulations—adds another layer of difficulty [7]. Approaches such as *domain randomization* and *observation design* seek to bridge this gap by increasing robustness and leveraging existing knowledge, respectively. However, these solutions still rely heavily on human expertise and trial-and-error efforts, demanding considerable time.

The incorporation of large language models (LLMs) into robotics offers a compelling way to lessen this human workload by tapping into their vast knowledge reserves. Previous studies highlight the potential of LLMs in areas such as code generation [8], robot action planning [9], and process improvement through feedback [10]. These efforts provide useful tools to cut down on manual labor and improve adaptability across diverse tasks. However, a fully automated framework that covers the entire pipeline—from design and training to deployment—has yet to emerge.

To address the problem discussed, we introduce *Anybipe*[1], an innovative framework that utilizes large language models (LLM) to automate the design, oversight and ongoing enhancement of robotic processes. Given a task description, *Anybipe* independently manages training, verification, tracking, and evaluation, eliminating the need for human involvement. By drawing on the deep knowledge embedded in LLMs, it adeptly handles intricate training challenges, even when task-specific prior data is scarce. The framework's structure is shown in Fig. 1. During training, we use a tailored RL algorithm with a reference policy that addresses the cold-start issue, integrates prior work, and speeds up convergence. For deployment and evaluation, we employ methods for data gathering and safety checks, paired with a new metric—the *homomorphic evaluator*—to measure the simulation-to-reality gap.

[1] Author Y. Yao, C. Gu, J. Du, F. Tan, Z. Zhu, and Jun-Guo Lu are with Department of Automation, Shanghai Jiao Tong University, Shanghai, China. Email: {godchaser, jiahengdu, wwddv1995, dyzz0928 , jglu }@sjtu.edu.cn, C. Gu: guchenyuFG@outlook.com

[2]W. He is with University of Michigan - Shanghai Jiao Tong University Joint Institute , Shanghai Jiao Tong University, Shanghai, China Email: goodmorning_hwt@sjtu.edu.cn

† The second to the fourth author contributed equally.

* is for corresponding authorship.

[1]https://github.com/sjtu-mvasl-robotics/AnyBipe.git

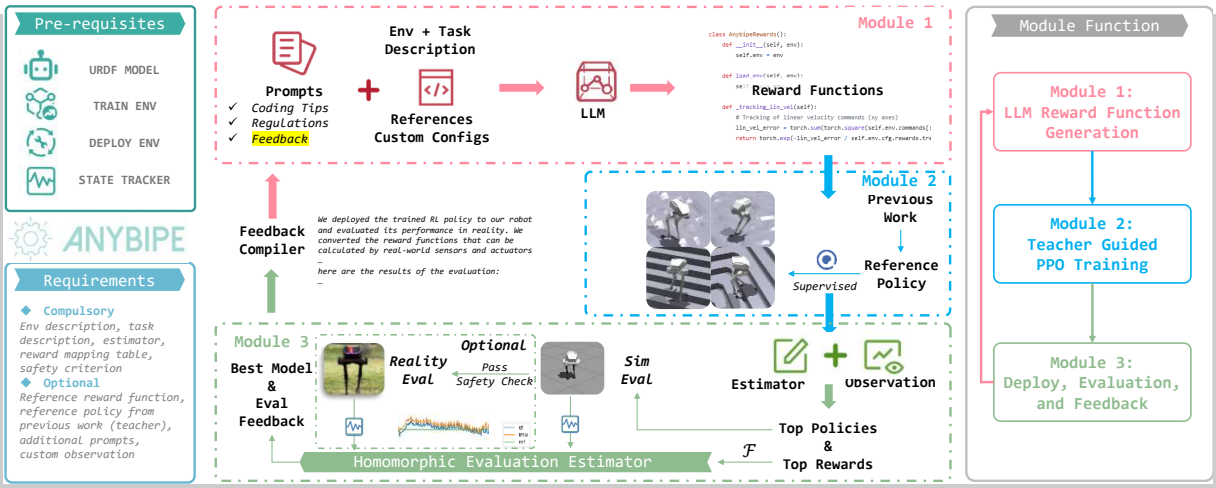


Fig. 1: Our framework is organized into three interconnected modules. After receiving prerequisites and requirements, it generates a reward function via LLM, trains it in simulation, and evaluates performance in both simulation and reality, offering key feedback. This process requires minimal human effort by automating all procedure with scripts.

We tested *Anybipe*'s capabilities on three bipedal robots with differing degrees of freedom (DoFs) and various initial prompt setups, confirming its versatility and ability to operate across diverse robots without human input. Our prompts produce over 60% successful task policies, climbing to 100% after a few iterations. The top policies outstrip human-designed solutions by as much as 33.3% in performance. In the guided RL segment, we see faster convergence, with success rates rising by an average of 52.1% and up to 94.4% through curriculum learning. Plus, our *homomorphic evaluator* consistently aligns simulation and real-world results, effectively gauging the simulation-to-reality gap.

The rest of this paper is organized as follows: section III explains the design principles and implementation of *Anybipe*'s modules. section IV covers the experiments we ran to verify each module's role and show the framework's training and deployment strengths. Finally, section V wraps up our contributions and sketches out future research paths.

II. Related Works

Reinforcement Learning for Bipedal Robots. Reinforcement learning (RL) has achieved significant success in enabling legged robots to learn locomotion and motion control through data-driven methods, allowing them to adapt to diverse environmental challenges [11]–[13]. However, prior works suffer from inefficient exploration in high-dimensional action spaces, leading to prolonged cold-start phases. Our framework propose a semi-supervised RL framework within the Isaac Gym environment that integrates reference policy, anchoring initial policy exploration near stability-proven actions from traditional controllers, while enforcing safety constraints via ground-truth-aligned reward terms. This dual mechanism of directed exploration and constraint-aware learning significantly accelerates policy initialization without compromising the discovery of novel locomotion strategies.

Large Language Model Guided Robotics. Large language models (LLMs) have demonstrated considerable capabilities in task understanding [14], semantic planning [15], and code generation [16], [17], allowing them to be effectively integrated into a variety of robotic tasks. LLMs automate environmental analysis [18], [19], reward functions design [20], [21], and task-action mapping [22], [23]. However, challenges such as data scarcity, real-time performance, and real-world integration remain [24]. In addition, due to the lack of direct perception of actual data, tranditional LLM-driven code design struggle to effectively incorporate real-world feedback and requires human feedback. Our framework addresses this by adopting environmental characteristics and safety constraints as priors, and uniquely combines homomorphic feedback from real-world applications, reducing the need for human feedback in the process.

Sim-to-real Training and Deploying Techniques. The gap between simulated environments and real-world conditions, known as the "reality gap", presents significant challenges for deploying RL strategies in robotics [25], [26]. Techniques such as domain randomization [27]–[29] and system identification [30]–[32] are widely used to address this issue. Researchers have proposed sim-to-real solutions for bipedal robots to handle tasks such as turning and walking [33], [34]. Recent work has also integrated LLMs to enhance environmental modeling and reward function design, making simulations more reflective of real-world complexity [10]. However, most approaches still rely on separate training in simulation and real-world evaluation, often using human feedback to assess sim-to-real effectiveness. Our work extends these techniques by introducing an evaluation loop that continuously monitors sim-to-real performance during deployment. Through the Homomorphic Evaluator and sim-to-real gap quantification metrics, we successfully achieved automatically generated real-world feedback.

III. METHODS

In this section, we introduce the *AnyBipe* framework in detail. The framework consists of three interconnected modules designed to enhance reward design, facilitate semi-supervised RL training, and automate evaluation and feedback. Users are required to provide a URDF model, a task description, an RL environment, and an operational control platform such as ROS [35]. Additionally, a runtime data collector, known as the State Tracker, is essential for gathering critical data, including IMU readings, joint positions, velocities, and torques. Leveraging a well-engineered prompt framework, we enable LLMs to generate suitable reward functions based on user-provided patterns, ensuring both functionality and safety. These reward functions are then trained in modified RL environments, aided by cold start techniques to improve convergence. The top-performing policies are validated using a homomorphic estimator to assess safety and compare performance with the training stage. All results are compiled into feedback prompts to guide iterative improvements. We model our guided training process as a Markov Decision Process (MDP) defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathrm{R}, \pi_{ref}, \beta)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the training environment, R is the reward function, π_{ref} is the reference policy, and β controls its influence. The procedural steps are outlined in Algorithm 1.

A. Module 1: LLM Guided Reward Function Design with Proper Prompt Engineering

We build on the Eureka algorithm's reward function design [20], which uses predefined environmental and task descriptions $\mathcal{D}(\mathcal{T})$ to enable LLMs to autonomously refine reward functions. However, initial usability issues often necessitate multiple iterations for functional training code, and Eureka tends to miss discrepancies between training $\mathcal{T}$ and real environments $\mathcal{R}$, yielding computationally costly, less effective reward functions that risk undefined behaviors [36]. Safety considerations for tasks like bipedal movement are also insufficient, despite efforts with Reward-Aware Physical Priors (RAPP) and LLM-led Domain Randomization [10].

To tackle these issues, we developed a robust context establishment mechanism to help LLMs create reliable, secure reward functions. Our iterative ReAct (Reason + Act) framework [37] refines coding outputs using real-world performance. Prompts are structured in three phases: system prompt, task specification, and feedback. The system prompt stage generates knowledge prompts for the RL environment and applies few-shot prompting [38] for reward function creation. Environmental variables (e.g., "root states", "feet height", "rigid body state") are cataloged in a reference table to avoid non-existent variables and enhance integration within $\mathcal{D}(\mathcal{T})$. For complex tasks like humanoid walking, users can add custom prompts via our "coding references" guide, enabling LLMs to learn from human examples and produce task-specific code. A "coding restrictions" section provides negative prompts, enforcing safety constraints like contact forces, DoF limits, and torque bounds. Experiments show LLMs effectively integrate these restrictions and variables, crafting highly efficient reward functions, demonstrating strong contextual tracking and directive adherence [39].

The task specification stage initially provides essential training details, including a robot description, key parameters (e.g., base height, average foot air time), and an introduction to reward-writing techniques. After the first iteration, this stage transitions to a comprehensive evaluation scheme, as detailed in subsection III-C, which serves as a feedback prompt. This prompt includes multiple components: the performance of the reward function, the consistency

Algorithm 1 AnyBipe Framework Process

Pre-requisites: Training environment $\mathcal{T}$, deployment environment $\mathcal{R}$, robot state tracker st
Require: Environment description $\mathcal{D}(\mathcal{T})$, prompt set p, training environment estimator $\mathfrak{E}_{train}$, homomorphic estimator mapping function $\mathcal{F}$, safety evaluation criterion SA. MDP $\mathcal{M}$, LLM model LLM, feedback prompt compiler COMPILE
Optional: Human-engineered reward function $\mathbf{R}_{ref}$, reference policy (previous work) π_{ref}, additional prompts p_{add}, critical human factor to be observed obs_c
Hyperparameters: Iteration N, number of reward candidates K, best sample percentage c_{bs}, teacher model coefficient β, environment estimator coefficient c_e, human factor observation coefficient c_{obs}
Input: Task description $\mathcal{D}$
$\mathbf{R}_{ref} \leftarrow$ **if** pre-defined **then** reference reward **else** None
$\pi_{ref} \leftarrow \mathrm{RL}(\mathbf{R}_{ref})$ or user-defined **if** exists **else** None
for $i \leftarrow 1$ **to** N **do**
    // Module 1: Reward Function Generation
    $p_{in} \leftarrow p + p_{add} + p_{feedback}$
    $\mathbf{R} \leftarrow \mathrm{LLM}(\mathcal{D}, \mathcal{D}(\mathcal{T}), p_{in}, \mathbf{R}_{ref})$
    // Module 2: Semi-Supervised RL Training
    $\Pi, \mathrm{Obs} \leftarrow \mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathrm{R}, \pi_{ref}, \beta)$
    // Module 3: Automated Evaluation and Feedback
    $n_{bs} \leftarrow \lceil c_{bs} \cdot K \rceil$
    $p_{feedback} \leftarrow$ None
    $Criterion \leftarrow c_e \cdot \mathfrak{E}_{train}(\Pi) + \sum c_{obs} \cdot \mathrm{Obs}$
    $\mathbf{R}_{bs}, \pi_{bs} \leftarrow \arg\max_{Criterion}(\mathbf{R}, \Pi)$
    $\hat{\mathbf{R}}_{bs} \leftarrow \mathcal{F}(\mathbf{R}_{bs})$
    for all $\pi, \hat{\mathbf{R}}$ **in** $\pi_{bs}, \hat{\mathbf{R}}_{bs}$ **do**
        $\pi_{real} \leftarrow (\mathcal{T} \rightarrow \mathcal{R})(\pi)$
        $\mathfrak{E}_{sim} \leftarrow \mathrm{EVAL}_{sim}(\hat{\mathbf{R}}(st(O)), \pi_{real})$
        $p_{feedback} += \mathrm{COMPILE}(p, \mathfrak{E}_{sim})$
        if $SA(\mathfrak{E}_{sim})$ **is** True **then**
            $\mathfrak{E}_{real} \leftarrow \mathrm{EVAL}_{real}(\hat{\mathbf{R}}(st(O)), \pi_{real})$
            $p_{feedback} += \mathrm{COMPILE}(p, \mathfrak{E}_{real})$
        end if
    end for
    $\mathbf{R}_{ref}, \pi_{ref} \leftarrow \arg\max_{Criterion}(\mathbf{R}_{bs}, \pi_{bs})$
    $p_{feedback} += \mathrm{COMPILE}(p, \pi_{ref})$
end for
Output: Best policy π, best reward function $\mathbf{R}$

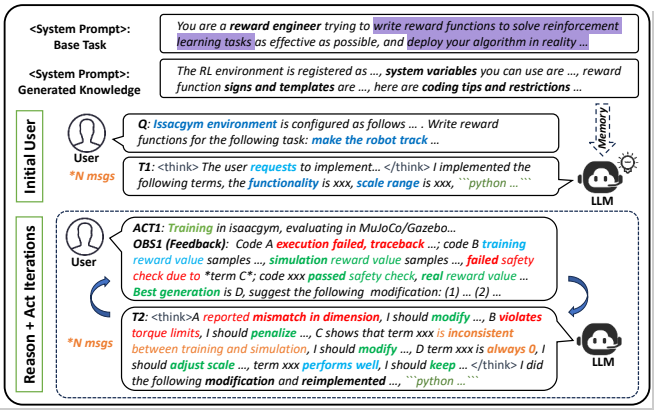


Fig. 2: Demonstration of LLM reward generation iterations. LLMs use a CoT format, adjusting outputs based on feedback. Each iteration produces $N = K$ reward function samples, which are trained, evaluated, and the best is used as a template for the next iteration.

of the homomorphic estimator with the reward functions, and adherence to safety restrictions. To ensure alignment, the evaluator incorporates pre-defined human preferences, such as human factor observations (obs_c) and user-defined observation terms labeled with "gt" for consistency across LLM-generated reward functions. The success rate is also tracked to identify the best-performing sample, which is then used to guide the LLM in adjusting reward terms and coefficients. Additionally, self-consistency techniques [40] assist the LLM in evaluating individual reward terms, determining their effectiveness, and making necessary modifications. For non-executable code, error logs related to the reward file are extracted and provided as feedback, along with adjustment tips to refine the reward function further.

To enhance the generation process, we guide the LLMs to produce reward functions in a Chain-of-Thought (CoT) format [41], which includes step-by-step explanations of their implementation. This approach encourages the LLMs to think through the logic of the reward function, making it easier to identify and resolve issues. The generated code is instructed to be presented within a code block at the end of the output, which is then processed by our script to remove $< think >$ tags and extract the target code. Our experimental results demonstrate that the CoT method significantly improves the LLM's ability to detect and address issues within the code, leading to more reliable and effective reward functions.

B. Module 2: RL Training with Reference Policy for Cold-Start

In the initial training phase, LLMs often produce poorly scaled reward terms without external feedback, leading to slow convergence—a challenge known as the "cold start" problem. Integrating a reference policy provides guidance, helping the policy network converge toward the target objective. This section outlines modifications to direct reinforcement learning (RL) training, leveraging Legged Gym and the Proximal Policy Optimization (PPO) algorithm [42] as the foundation. We assume a baseline policy, π_{ref}, derived from functional policies, with interfaces for three teacher inputs: traditional control methods (e.g., Model Predictive Control or Whole Body Control), pre-trained policies (PyTorch or ONNX), or simple locomotion distributions like sine waves.

To enhance PPO, we adjust the clip function loss as follows:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) + \beta\,\text{KL}[\pi_{\text{ref}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]], \tag{1}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio, $\hat{A}_t$ estimates the advantage at time t, ϵ is a small positive constant, and β weights the KL divergence between the reference and PPO policies. Here, $\hat{\mathbb{E}}_t$ denotes expectation over time steps.

This modification ensures similarity between the trained and reference policies. Although integrating π_{ref} into PPO's probabilistic framework is complex, we approximate it as a Dirac distribution due to the deterministic nature of actions a_t for a given state s_t and previous action a_{t-1}. With sufficient observations, the KL divergence $\hat{\mathbb{E}}_t\left[\text{KL}(\pi_{\text{ref}}, \pi_\theta)\right]$ simplifies to:

$$\frac{1}{N}\sum_{i=1}^{N}\left[\log(\sqrt{2\pi\sigma_{\theta,i}^2}) + \frac{(a_{\text{ref}} - \mu_{\theta,i})^2}{2\sigma_{\theta,i}^2}\right]. \tag{2}$$

Here, $\mu_{\theta,i}$ and $\sigma_{\theta,i}^2$ represent the mean and variance of the policy π_θ at step i, and a_{ref} is the reference action. This approximation holds with adequate data, and β should be small to guide the policy away from local minima without limiting exploration.

In the *AnyBipe* framework, we provide a template for integrating existing policies as teacher functions into RL training. If a human-engineered reward is set as ground truth, it is trained first, and the resulting policy becomes the teacher. Users can also supply custom policies via the template. After each iteration, the reference policy updates to the best-performing trained policy, provided it passes a safety check. This setup compares human-engineered and LLM-generated policies, offering insights into metric accuracy and preventing degradation by monitoring reward term behavior post-modification.

C. Module 3: Feedback From Simulation and Deployment Stage With Minimal Human Effort

Traditional Eureka-like algorithms collect training feedback only for the next generation, leaving the generated policies to engineers for examination and deployment. In contrast, our framework automates this process using Python, Bash, and C++ scripts, implementing a robust safety check and providing a numerical measure for the sim-to-real gap, called the "homomorphic estimator". This estimator enables LLMs to identify key terms that impact deployment.

To evaluate policy performance, we define a set of consistent observations with "ground-truth" (gt) labels across

all generations. For our experiments, this set includes linear velocity, angular velocity, and survival time. The product of these indicators, termed the *environment estimator* $\mathfrak{E}_{train}$, measures policy success. Users can also define custom observation/reward factors, termed human factors obs_c, with corresponding weights $c * obs$. The weighted sum of these factors is used to compute the total score, ranking policies for simulation and deployment validation.

$$\mathbf{R}_{bs}, \Pi_{bs} = \arg\max_{n_{obs}} c_e \cdot \mathfrak{E}_{train}(\Pi) + \sum c_{obs} \cdot \mathrm{Obs}_c \,. \quad (3)$$

This procedure transforms reward functions into new estimators that maintain consistency between training and simulation, providing a quantitative measure of the sim-to-real gap and helping the LLM refine its approach.

To ensure real-world safety, selected policies must pass a safety check, denoted as SA, before being compiled by ROS. Users must implement SA using the provided template, which evaluates simulation data such as torques, positions, and ground contact forces. If all checks pass, SA returns true, and the script automatically compiles and executes the reality deployment code. If any checks fail, the script identifies the failed terms and generates a feedback prompt. The reality evaluator works similarly to the simulation evaluator, collecting and evaluating data. Users can monitor the process and stop deployment if dangerous behavior is detected. External stops are recorded, indicating deployment failure; otherwise, deployment is considered successful. These steps provide a score contributing to policy selection, as follows:

$$R_{best}, \pi_{best} = \arg\max \left[[\mathcal{F}(\mathbf{R}_{bs})_{sim} + \mathcal{F}(\mathbf{R}_{bs})_{real}](\mathcal{T} \to \mathcal{R})(\Pi_{bs}) \right]. \quad (4)$$

Except for the final reality deployment stage, which may require supervision (though it can operate autonomously or with Vision-Language Models (VLMs) and cameras substituting human functions), the entire cycle runs autonomously. Evaluation, safety checks, and deployment procedures are human-free. Even initial reference rewards or pre-existing policies are optional, as LLMs show strong potential in handling zero-shot generation tasks [43]. This framework offers a method for tuning reward functions for robot engineers and supports developing RL policies from scratch for robots without prior RL implementations.

IV. Experiments

A. Experimental Setup

Experiments were conducted with three bipedal robots: the 6 DoF pointfoot robot P1 from Limx Dynamics, the 10 DoF humanoid Unitree H1 (lower body only), and the 29 DoF humanoid from Turin Robots (12 lower body DoF, with other DoF fixed during training), as shown in Fig. 3. Each robot was equipped with different initial prompt sets: P1 used human-engineered reward functions as ground truth for comparison; Unitree H1 used human-engineered functions as templates; and Turin, without prior RL implementation, used templates partially adapted from the open-source humanoid-gym project, with no ground-truth reward or reference policy. LLMs used include OpenAI GPT-4o, Anthropic Claude-3.5-Sonnet, and Deepseek-R1 to validate the versatility of our prompt groups and scripts. The framework was trained on a GPU server with 4 NVIDIA RTX 3090 GPUs. The robots were trained on common and trimesh terrains, with task settings including $N = 5$ iterations, $K = 16$ samples, and the top 15% (3 best samples) selected for each task. Thanks to the GPU allocator algorithm, a total of 80 training runs were completed in 8 hours, including LLM generation and evaluation time. *To demonstrate minimal human effort, the entire experiment was conducted autonomously by the framework, with only the generated policies and checkpoints used to reproduce the results in figure form.* For the two humanoids, the framework used only *simulation* evaluation with SA feedback. For P1, real-world deployment testing validated the auto-deployment process. The GPU server was connected to robot controllers, and simulation environments (MuJoCo mjviewer and Gazebo) operated in headless mode.

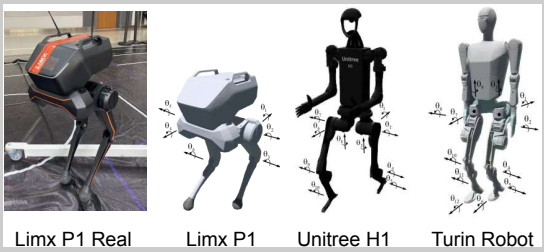


Fig. 3: Robot and DOF definitions.

Table I lists general *environment estimators* as reward functions. These estimators maintain the same form throughout training and evaluation.

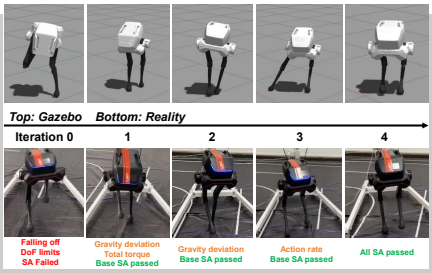
Reward Name	Expression Form
Survival	$R_{surv} = \int_0^{t_{\text{term},i}} dt$
Tracking Linear Velocity	$R_{vel} = \exp(-\frac{\lVert v - v_{ref} \rVert^2}{\sigma_l^2})$
Tracking Angular Velocity	$R_{angl} = \exp(-\frac{\lVert \omega - \omega_{ref} \rVert^2}{\sigma_a^2})$
Success	$R_{succ} = R_{surv} \cdot R_{vel} \cdot R_{angl}$

TABLE I: Examples of important rewards

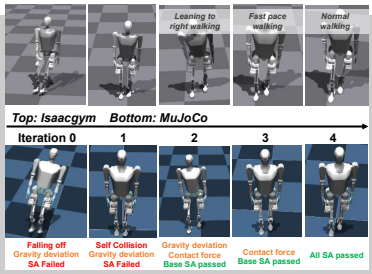
B. Module Analysis

Before introducing the outcome for the whole framework process, we first conduct several experiments seperately on each module to examine its effectiveness. We evaluated the LLM performance, RL-training performance, homomorphic performance, and whether SA criterion functions normally.

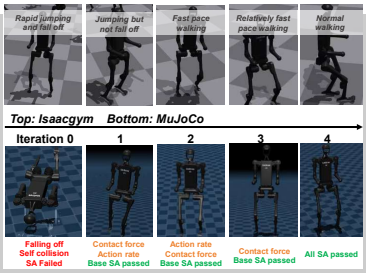
We evaluated each LLM base by testing prompt design and various augmentation. The temperature for generation is set to **0.4** for all models. We define success rate (number of code that can be executed for all generation, SR) and max normalized reward success (max $R_{succ}/num\_steps\_per\_env$, the average on three experiments, Max S.) as two important



P1 iterations.



H1 iterations.



Turin iterations.

Fig. 4: Best generation policies for 5 iterations evaluated in training and evaluating environments. For P1, it is evaluated also in reality. Figures indicate the policy improvement among iterations and safety check SA feedback below each iteration. Colors indicate the severeness of different SA indicators.

indicators. The test result among the LLMs are shown as follows. We noticed that Deepseek-R1 and Claude-3.5-Sonnet shows best performance.

Name	P1	H1	Turin	SR	Max S.
GPT-4o	228/240	173/240	52/240	62.9%	0.824
Claude-3.5	232/240	213/240	201/240	89.7%	1.032
Deepseek-R1	-/-	232/240	211/240	92.3%	1.060

TABLE II: Number of run-able code / total generation on different bot tasks, success rate, max normalized reward success for different LLMs.

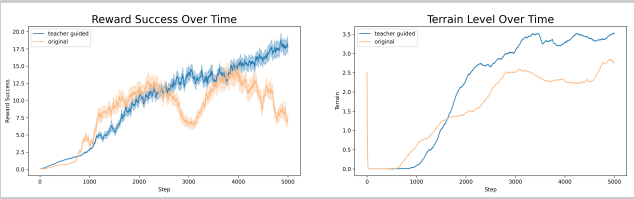


Fig. 5: Reward success, terrain level for teacher guided and original RL training with human-engineered rewards, using P1 as example. Blue line shows trending of training with teacher, and orange one is pure PPO.

To show the strength of reference policy, we take the training of P1 as example, set the reference policy using human-engineered reward function trained policy (5000 epochs training, flat terrain), and compare the training result using same reward function under trimesh terrain. The teacher coefficient is set to $\beta = 0.5$ with curriculum learning on. Performance was measured using *reward success* and *terrain level*[2] . Results in Fig. 5 showed that the teacher-guided model had more stable training, with faster and less volatile reward growth.

For the Homomorphic evaluation part, the following table shows some evaluation indicators before and after the conversion for P1 training, and the evaluation results under the best model. It can objectively reflect some differences from simulation to reality. Each column seperately represents reward function name, reward in isaac gym, homomorphic measurement in gazebo, in reality, and the mapped tracking result in reality (30 seconds of tracking, where env configuration requires at least 1.6s tracking).

Name	Gym	Gazebo	Reality	Mapping (real/targ)
Track lin vel	0.8893	0.9138	0.8504	0.86 (1.0) m/s
Track ang vel	0.7652	0.6500	0.6013	0.06 (0.10) rad/s
Feet distance	-0.31	-0.00	-0.00	$>$ 0.1m ($>$ 0.1m)
Standing still[1]	-5.35	-6.16	- 11.20	5.8 (30) s
Survival time	0.86	1.0	1.0	30 (30) s

Note: [1]Standing still requires robot to maintain speed 0 for certain time.

TABLE III: Examples of homomorphic evaluation for point-foot robot.

For the safety check criterion SA, we define two levels: the **base** criterion, which includes falling off, self-collision (not implemented for P1[3]), and violating DoF and torque limits; and the **warning** criterion, which includes high feet contact forces (not implemented for P1), high action rate, high total torque, and large gravity projection deviation. Policies passing the base safety tests are considered deployable, and those without warnings are regarded as safe. We conducted 3 training groups for each robot, generating 240 samples per robot (5×16 results per run). Models passing the base test are labeled as positive, and those passing both tests as strict positive. The ground truth is established by manually evaluating each policy's behavior in simulation to identify true positives and negatives. We then compute the average precision (AP) and average recall (AR) to assess SA performance. We also list another success rate (SR) defined as the percentage of policy passed the safety check. Evaluation proved that properly implemented SA can identify whether policy is safe under most scenarios. This also shows that prompts with better initial prompts leads to higher success rate.

C. Framework Analysis

The experimental setup is summarized in Table VI. For each type of experiment, we present results using different

[2]Terrain level is a curriculum monitor for average terrain heights, regarding to trimesh and stair cases defined in Isaacgym.

[3]Contact detection extraction is implemented in MuJoCo only

Name	Iter 0	Iter 1	Iter 2	Iter 3	Iter 4	Human Engineered
P1 Pointfoot (iter=5000)	14.7 (5.5)	7.1 (6.3)	9.2 (8.7)	15.5 (11.2)	25.6 (20.8)	19.2 (19.2)
Unitree H1 (iter=2000)	0.0097 (0.0035)	21.2 (12.4)	25.2 (21.7)	25.0 (22.4)	25.8 (24.9)	23.7 (23.7)
Turin Robot (iter=2000)	10.2 (1.96)	52.3 (42.9)	58.1 (51.8)	66.3 (55.4)	67.0 (57.8)	None (None)

TABLE IV: Reward success $\mathfrak{E}_{train}$, over iterations compared to human engineered ones in format of batch max (batch average). The upper bound of the term is restrained by $num\_step\_per\_env$.

Name	AP	AR	S-AP	S-AR	SR (S-SR)
P1 Pointfoot	94.3	96.4	100.0	92.6	58.7 (26.2)
Unitree H1	100.0	97.5	93.3	84.0	65.4 (18.8)
Turin Robot	97.6	96.1	83.8	93.9	52.9 (15.4)

TABLE V: Precision and recall for safety criterion in percentage form, 's' label means strict indicator.

(a) Restrained (b) Carpet (c) Hard Ground (d) Soft Ground (e) Stairs

Fig. 6: Reality experiments for P1 conducted on different terrains, adopting *AnyBipe*'s best policy.

LLM bases[4]. Detailed results and setup for similar experiments can be found on GitHub.

Name	LLM	Train Iter	Sim Env	Real Env
P1 Pointfoot	GPT-4o	5,000	Gazebo	✓
Unitree H1	Claude-3.5	2,000	MuJoCo	×
Turin Robot	Deepseek-R1	2,000	MuJoCo	×

TABLE VI: Framework experimental set-up.

Fig. 4 describe the best training result iterating over framework iterations, along with SA feedback indicating different levels of safety violation (red for not safe, yellow for potential problem warning, and green for passed check). P1 passes all SA check and all reality deployments are listed. The other two are listed in training-simulation pairs. These experiments prove that *AnyBipe* is capable of guiding LLM to realize what problems might occur in certain reward function implementation, and can act correspondingly to solve the problem. By iterations the unsafe problems are properly handled while maintaining code set effective.

We visualize the reward and terrain level curves for P1 training, comparing them to human-engineered reward curves extracted from TensorBoard logs in Fig. 6. For the three experiments, we report the average success rate at the midpoint and end of training for 5 iterations, comparing them with human-engineered metrics (considered state-of-the-art, as they are available open-source) in Table IV. Our results demonstrate that *AnyBipe* can identify more effective reward function combinations after several iterations without human intervention. The framework can also autonomously explore suitable code implementations in zero-shot scenarios when no reference is provided (as shown in the Turin task). This highlights the framework's potential for training newly designed robot configurations.

Experiments in Fig. 6 indicates that the trained policy can be truly deployed in reality, and maneuvers different kinds of terrain types.

[4]The choice of LLMs varies for different robots. P1 showed promising results with GPT-4o, and alternative models did not significantly improve performance. We intended to use Deepseek-R1, but its official API was unstable during the experiment (Jan. 2025 to Feb. 2025), so we opted for Claude-3.5-Sonnet instead.

V. Conclusion

AnyBipe has presented an end-to-end framework for training and deploying bipedal robots, leveraging state-of-the-art Large Language Models (LLMs) to design reward functions tailored to specific tasks. The framework has provided interfaces that allow users to supply coding references and integrate pre-existing models to facilitate the training process. Furthermore, it has incorporated feedback from both simulated and real-world test results, enabling the execution of Sim-to-Real tasks without human supervision while also offering improvement directions to the LLMs. We have validated the effectiveness of each module, as well as the system's capacity to guide the robot in learning locomotion in complex environments, progressively improving the model by either creating new reward functions from scratch or refining existing ones. Moreover, this framework has demonstrated potential for transferability to other robotic task planning scenarios.

Despite these strengths, there remain areas for further improvement. Treating observations and reward terms solely as human factors may not fully capture human preferences, and visual factors should also be considered. Also, the framework automates the whole training process, but still requires humans to implement the basic RL environment following legged gym and our template, not generating environment directly from LLMs. Furthermore, incorporating VLMs into the safety criterion could improve the precision of judgment.

Future work will focus on advancing the framework in key areas: first, expanding its application to a broader range of robotic tasks to assess its generalizability, testing its effectiveness beyond locomotion; second, introducing a supervisory process during the training stage to allow the framework to autonomously determine the duration of reinforcement learning training, rather than relying on a fixed total length; and third, refining the model evaluation process by incorporating visual and conceptual feedback to achieve a more comprehensive state estimation.

REFERENCES

[1] B. Acosta, W. Yang, and M. Posa, "Validating Robotics Simulators on Real-World Impacts," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6471–6478, 2022.

[2] E. Šutinys, U. Samukaitė-Bubnienė, and V. Bučinskas, "Advanced Applications of Industrial Robotics: New Trends and Possibilities," *Applied Sciences*, vol. 12, no. 1, p. 135, 2022.

[3] F. Ciardo and A. Wykowska, "Humanoid robot passes for human in joint task experiment," in *Science Robotics*, vol. 7, no. 68, 2022.

[4] A. Lobbezoo and H.-J. Kwon, "Simulated and Real Robotic Reach, Grasp, and Pick-and-Place Using Combined Reinforcement Learning and Traditional Controls," *Robotics*, vol. 12, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2218-6581/12/1/12

[5] D. Ernst and A. Louette, "Introduction to reinforcement learning," *Business & Information Systems Engineering*, vol. 66, no. 1, pp. 111–126, 2024.

[6] K.-C. Hsu, A. Z. Ren, D. P. Nguyen, A. Majumdar, and J. F. Fisac, "Sim-to-Lab-to-Real: Safe reinforcement learning with shielding and generalization guarantees," *Artificial Intelligence*, vol. 314, p. 103811, 2023.

[7] Salvato, Erica and Fenu, Gianfranco and Medvet, Eric and Pellegrino, Felice Andrea, "Crossing the Reality Gap: A Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning," *IEEE Access*, vol. 9, pp. 153 171–153 187, 2021.

[8] N. Shah, Z. Genc, and D. Araci, "Stackeval: Benchmarking llms in coding assistance," in *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[9] W. Xia, D. Wang, X. Pang, Z. Wang, B. Zhao, D. Hu, and X. Li, "Kinematic-aware Prompting for Generalizable Articulated Object Manipulation with LLMs," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 2073–2080.

[10] Y. J. Ma, W. Liang, H. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman, "DrEureka: Language Model Guided Sim-To-Real Transfer," in *Robotics: Science and Systems (RSS)*, 2024.

[11] J. Hwangbo, J. Lee, L. Wellhausen, H. Kolvenbach, and M. Hutter, "Learning Agile and Dynamic Motor Skills for Legged Robots," *Science Robotics*, vol. 4, no. 26, p. Art. no. eaau5872, 2019.

[12] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Learning, planning, and control for quadruped locomotion over challenging terrain," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 236–241.

[13] J. Lee, J. Hwangbo, L. Wellhausen, H. Kolvenbach, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[14] W. Huang, N. Gopalan, M. Ahn *et al.*, "Inner Monologue: Embodied Reasoning through Planning with Language Models," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[15] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, *et al.*, "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[16] Y. Wang and H. Li, "Code completion by modeling flattened abstract syntax trees as graphs," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 16, 2021, pp. 14 015–14 023.

[17] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500.

[18] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang, "Gensim: Generating robotic simulation tasks via large language models," *arXiv preprint arXiv:2310.01361*, 2023.

[19] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, K. Fragkiadaki, Z. Erickson, D. Held, and C. Gan, "Robogen: Towards unleashing infinite data for automated robot learning via generative simulation," *arXiv preprint arXiv:2311.01455*, 2023.

[20] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv: Arxiv-2310.12931*, 2023.

[21] 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.

[22] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu *et al.*, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023.

[23] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *IEEE Access*, 2024.

[24] F. Zeng, W. Gan, Y. Wang, N. Liu, and P. S. Yu, "Large language models for robotics: A survey," *arXiv preprint arXiv:2311.07226*, 2023.

[25] A. Brunnbauer *et al.*, "Latent Recovery for Sim-to-Real Transfer in Robot Reinforcement Learning," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1700–1706.

[26] J. Betz and H. Zheng, "Bypassing the Simulation-to-Reality Gap: Online Reinforcement Learning Using a Supervisor," in *2023 21st International Conference on Advanced Robotics (ICAR)*. IEEE, 2023, pp. 325–331.

[27] F. Muratore, F. Ramos, G. Turk, W. Yu, M. Gienger, and J. Peters, "Robot learning from randomized simulations: A review," *Frontiers in Robotics and AI*, vol. 9, p. 799893, 2022.

[28] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 3803–3810.

[29] G. Tiboni, P. Klink, J. Peters, T. Tommasi, C. D'Eramo, and G. Chalvatzaki, "Domain Randomization via Entropy Maximization," *arXiv preprint arXiv:2311.01885*, 2023.

[30] K. Åström and P. Eykhoff, "System identification-A survey," *Automatica*, vol. 7, no. 2, pp. 123–162, 1971.

[31] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 465–472, 2011.

[32] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[33] H. Duan, A. Malik, J. Dao, A. Saxena, K. Green, J. Siekmann, A. Fern, and J. Hurst, "Sim-to-Real Learning of Footstep-Constrained Bipedal Dynamic Walking," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 10 428–10 434.

[34] F. Yu, R. Batke, J. Dao, J. Hurst, K. Green, and A. Fern, "Dynamic Bipedal Turning through Sim-to-Real Reinforcement Learning," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, 2022, pp. 903–910.

[35] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[36] Y. Kim, H. Oh, J. Lee, J. Choi, G. Ji, M. Jung, D. Youm, and J. Hwangbo, "Not only rewards but also constraints: Applications on legged robot locomotion," *IEEE Transactions on Robotics*, 2024.

[37] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[38] 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.

[39] 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.

[40] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[41] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv: Machine Learning*, vol. abs/1707.06347, 2017.

[43] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.