Self-Evolving Meta-Skill Learning for Humanoid Robots with Large Language Models

Xinyuan Luo
Mechanical Engineering and Material Science
Duke University
Durham, NC 27705, USA
xl521@duke.edu

Ajay Vikram Periasami
Computer Science
Duke University
Durham, NC 27705, USA
ajay.vikram@duke.edu

Abstract

Humanoid robots require flexible motor skills and the ability to acquire new behaviors to operate in long-horizon, unstructured tasks. However, reinforcement learning pipelines are heavily constrained by manually designed reward functions, which do not scale as tasks or skill libraries grow. This project presents a self-evolving, LLM-guided skill learning framework in which a language model decomposes high-level tasks, identifies required skills, and synthesizes reward functions for missing primitive or combined skills. The system closes the loop by evaluating each generated reward through RL training and using performance feedback to refine the reward iteratively without human intervention. To assess the reliability and generalization of LLM-generated rewards, we benchmark multiple state-of-the-art language models and compare their performance against ground-truth reward definitions. We additionally evaluate the LLMs' ability to produce learnable rewards for new skills such as jumping, kicking, bending, and side-stepping as well as composite multi-skill behaviors. Overall, the results highlight the potential of LLM-driven reward synthesis as a scalable approach to autonomous skill acquisition in humanoid agents. Our code is publicly available at https://github.com/ajay-vikram/SkillBlenderLC.

1 Introduction

Humanoid robots are increasingly expected to operate in unstructured environments, collaborate with humans, and execute complex behaviors that unfold over long time horizons. These capabilities require not only low-level motor control but also the ability to sequence diverse behaviors, adapt to novel situations, and acquire new skills when existing ones are insufficient. Reinforcement learning (RL) is a natural paradigm for training such behaviors, as it mirrors the trial-and-error learning process observed in biological agents. However, RL in robotics remains fundamentally constrained by the need for carefully hand-designed reward functions. Crafting these rewards requires deep domain expertise, extensive iteration, and is often brittle: a small change in environment dynamics or task structure can invalidate previously tuned reward terms. This makes manual reward engineering one of the central bottlenecks in scalable robot learning.

Although libraries of primitive skills can reduce some of this burden, they introduce another problem: a fixed set of skills cannot support open-ended task execution. Many long-horizon tasks require abilities not present in the initial skill set, such as jumping, balancing under perturbations, or coordinated whole-body actions. In traditional pipelines, introducing such new skills requires additional human input to specify motion objectives, write reward terms, and tune shaping coefficients. As tasks grow in complexity, the need for continuous human supervision becomes impractical.

At the same time, Large Language Models (LLMs) have demonstrated remarkable proficiency in structured reasoning, program synthesis, planning, and generating formally constrained code. These abilities make LLMs attractive tools for bridging high-level task

specifications and low-level controllers. However, existing uses of LLMs in robot learning often focus on either decomposing tasks into subtasks or generating reward functions in isolation. They do not close the loop: the LLM does not learn from the downstream performance of the RL agent, nor does it iteratively improve its own outputs based on actual training feedback. As a result, LLM-generated rewards may look reasonable in text but still fail to produce stable or learnable behaviors in practice.

This project proposes a self-evolving meta-skill learning framework designed to address these limitations. The core idea is to create an autonomous pipeline that can (1) decompose long-horizon tasks into manageable subtasks, (2) identify which skills are required, (3) detect when new skills must be created, (4) generate reward functions for those new skills, and (5) refine these rewards repeatedly until the RL agent successfully acquires the skill. This creates a closed-loop system in which the LLM is not only a generator of rewards but also an adaptive component that responds to real training performance.

By tightly integrating LLM reasoning, skill decomposition, reward synthesis, and RL-based feedback, this framework supports a form of autonomous lifelong learning in humanoid robots. It reduces dependence on manual engineering, enables scalable expansion of the skill library, and offers a promising pathway toward open-ended robotic intelligence. The long-term vision is a system that continuously improves, discovers new capabilities in response to task demands, and evolves without external supervision—mirroring the self-organizing nature of learning in natural organisms.

2 Related Work

Designing reward functions: Designing effective reward functions remains one of the most critical yet challenging aspects of reinforcement learning. Traditional manual reward shaping is often brittle and task-specific, while recent advances in LLMs have inspired a new research direction, leveraging natural language reasoning and code generation capabilities for automatic reward design.

Kwon et al. (2023) first explored this idea in Reward Design with Language Models, where they treated the LLM as a proxy evaluator translating natural language specifications of the task into executable reward signals. This work established the feasibility of using pretrained language models to bridge human intent and reinforcement learning objectives. Building upon this notion, Xie et al. (2024) proposed Text2Reward, a framework that enables data-free dense reward generation directly from textual task descriptions. Their system automatically synthesizes interpretable reward code and performs reward shaping without human intervention, providing a strong baseline for language-driven RL.

Beyond direct reward generation, several studies have leveraged LLMs for heuristic extraction and credit assignment. Bhambri et al. (2024) investigated extracting human-like heuristics from LLMs to construct auxiliary reward signals, demonstrating significant improvements in sample efficiency on sparse-reward tasks. Similarly, Qu et al. (2025) introduced LaRe, a language-guided latent reward redistribution framework that utilizes LLMs to reason over episodic trajectories and assign temporally consistent credit across sub-goals, effectively addressing long-horizon learning challenges.

Another emerging line of work focuses on iterative or evolutionary reward optimization with human feedback. Ma et al. (2024) introduced EUREKA, a human-level reward design algorithm that leverages coding LLMs to iteratively generate, evaluate, and refine executable reward functions through in-context evolutionary search. By combining environment-aware prompting and reward reflection, EUREKA achieves human-competitive or superior performance across a wide range of robotic and dexterous manipulation tasks. Similarly, Hazra et al. (2025) proposed REvolve, which integrates LLM-driven reward evolution with preference feedback to progressively refine reward functions. Sun et al. (2024) extended this idea through a cyclic automatic refinement framework (CARD), where the LLM repeatedly generates and evaluates reward code to maximize task performance. Both methods highlight the potential of coupling language reasoning with evolutionary optimization for scalable reward synthesis.

Task decomposition: Recent advances have also explored how LLMs can decompose complex natural-language instructions into executable skills or policies. Ahn et al. (2022) proposed SayCan, which combines LLM reasoning "Say" with affordance-based skill evaluation "Can", allowing robots to ground language instructions into feasible low-level actions. Liang et al. (2023) extended this idea in Code as Policies, where LLMs synthesize robot control programs directly from textual commands, composing perception APIs and control primitives to execute new tasks. Singh et al. (2022) introduced ProgPrompt, which provides a programmatic prompting structure that guides LLMs to generate grounded, context-aware action sequences rather than free-form text. Liu et al. (2025) developed DELTA, an LLM-driven framework that decomposes long-horizon tasks into subgoals using scene-graph representations and integrates them with classical task-and-motion planning. Huang et al. (2023) presented Instruct2Act, which maps multimodal instructions to executable robotic actions by generating end-to-end perception–planning–action code.

Overall, these studies collectively demonstrate that LLMs can serve as powerful priors for interpreting task intent, generating executable reward functions, and aligning reinforcement learning agents with human-desired behaviors. LLM-based planning further reveals their ability to decompose complex tasks into structured subgoals and select appropriate skills or policies for execution, enabling more general and scalable robot behaviors. Combined together, this method holds the promise of creating a lifelong learning agent capable of autonomously developing new skills over time. While previous studies have demonstrated that LLMs can generate effective reward functions, most of these evaluations have been limited to simplified or human-specified environments. Building upon these foundations, our work aims to investigate the planning and evaluative capabilities of LLMs within a more general and open-ended environment, focusing on how a humanoid robot can progressively evolve and acquire diverse skills.

3 Method

Our goal is to construct a closed-loop, self-evolving skill learning framework in which a humanoid robot can (i) interpret high-level natural-language tasks, (ii) decompose them into executable subtasks, (iii) identify missing skills, (iv) automatically synthesize reward functions for those skills, and (v) refine these rewards through reinforcement learning feedback until stable behaviors emerge. The overall system architecture is illustrated in Figure 1. We describe each module below.

3.1 Task Planner: LLM-Driven Skill Decomposition

Given a natural-language task description (e.g., *"Jump and then walk forward a few meters"*), the Task Planner queries a large language model to decompose the instruction into a sequence of actionable subtasks. The LLM is provided with (1) the current primitive skill library and (2) a structured prompt template to ensure that its output is formatted as a JSON list of subgoals and associated skills.

For each subgoal, the LLM compares the required motor capabilities with the existing skill library. If all skills necessary to accomplish the subgoal already exist, the Task Planner schedules them for execution. If the subgoal requires a capability absent from the library (e.g., *jumping*, *kicking*, *balancing*), it is flagged as a new skill that must be learned. This decomposition step allows the robot to bridge unstructured natural language with the skill-level control abstractions used by the underlying RL system. A concrete example of the structured decomposition output produced by the Task Planner is provided in Appendix A.1.

3.2 Reward Generator: LLM-Based Reward Synthesis

For every newly identified skill, the Reward Generator synthesizes an initial reward function using an LLM. The model is prompted with:

- the natural-language subgoal,

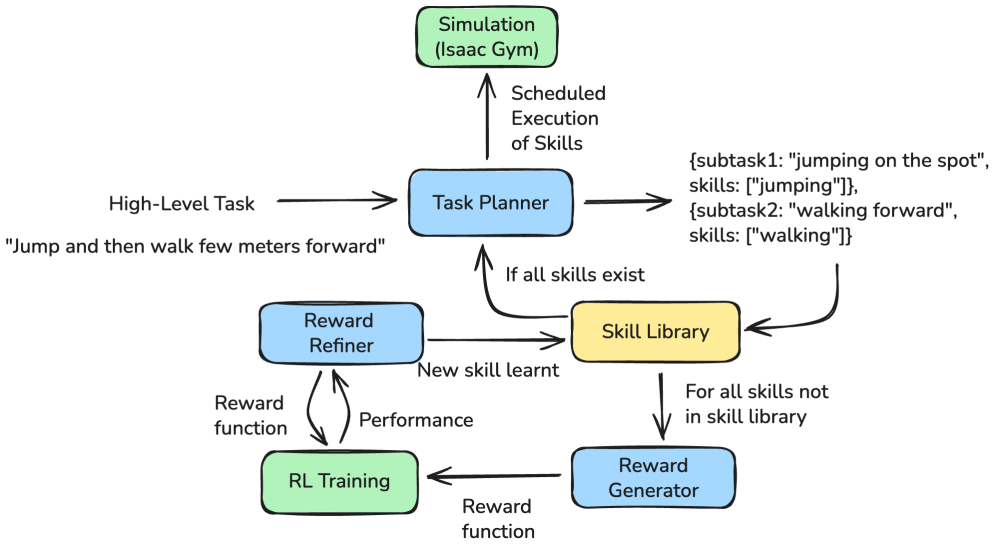


Figure 1: Overview of our self-evolving skill learning pipeline. High-level tasks are decomposed by an LLM into subtasks and skills. Missing skills trigger LLM-based reward synthesis, RL training, and iterative reward refinement. Successfully learned skills are added to the Skill Library for future reuse.

- example reward functions from existing primitive skills,
- environment state variables and dynamics from the simulation (extracted from Isaac Gym),
- constraints enforcing a consistent structure across all generated reward classes.

The LLM outputs a complete Python reward class that follows SkillBlender's reward API which is then passed directly to the RL training pipeline. This module enables zero-shot skill expansion, allowing the system to attempt behaviors it has never seen before.

3.3 Reinforcement Learning and Execution

Each generated reward function is deployed in NVIDIA Isaac Gym, where a PPO agent is trained to learn the corresponding behavior. The simulation provides rich proprioceptive feedback, and thousands of parallel environments accelerate training to practical scales.

If all required skills for a task already exist in the library, the Task Planner schedules them and executes the combined behavior directly. If new skills must be learned, they are trained first, stored, and then composed into longer-horizon behaviors.

3.4 Reward Refiner: Closing the Loop with RL Feedback

After initial RL training, performance metrics such as stability, joint torques, energy consumption, motion smoothness, distance-to-target and global task level metrics like success rate are passed back to the Reward Refiner module. The Reward Refiner prompts the LLM with the original generated reward code and numerical performance metrics. The LLM produces an improved reward function, which is retrained again in Isaac Gym. This iterative process continues until the skill reaches a performance threshold. Once validated, the newly learned skill is added to the global skill library, enabling future reuse.

This allows the robot to grow a progressively richer repertoire of motor primitives, which can be composed by the Task Planner when solving new tasks. Over time, the system evolves from simple primitives (e.g., walking, reaching) to complex behaviors (e.g., jumping, bending, kicking) through repeated reward generation and refinement.

4 Experiments

To evaluate our self-evolving skill learning framework, we conduct a series of experiments that examine the LLM's ability to (i) decompose tasks into meaningful primitive skills, (ii) generate reward functions for existing primitive skills, (iii) generate reward functions for novel primitive skills and (iv) handle long-horizon tasks. These experiments require a controlled and reproducible simulation environment in which both the generated rewards and learned policies can be systematically assessed, motivating the experimental setup described below.

4.1 Experimental setup

Following the framework SkillBlender proposed by Kuang et al. (2025), we implement our experiment in NVIDIA Isaac Gym (Makoviychuk et al. (2021)) with the PhysX physics engine.

Embodiments: SkillBlender supports three humanoid embodiments - the 19-DoF Unitree H1, the 21-DoF Unitree G1, and the 21-DoF Unitree H1-2. To simplify our experiments and improve training stability, we select the 19-DoF Unitree H1 as the primary embodiment for all evaluations.

Environments: SkillBlender provides four motion primitives and eight high-level tasks. For evaluating the LLM-generated reward values on existing skills, we adopt the squatting primitive as the base motion and the press-the-button task as the representative high-level behavior. To further examine the model's capability for new skill generation, we design a general-purpose environment named Basic, in which neither the reward function nor the task goal is explicitly defined. The environment exposes only a small set of primitive reward terms that are shared across all embodiments. In this setting, the LLM must (i) infer a task-specific reward formulation from the natural-language task description and (ii) determine all corresponding reward values for the selected reward terms.

Reinforcement learning: The observation space includes the robot's proprioceptive states and the target goal, and the action space covers all actuated joints. We train our policies using the Proximal Policy Optimization (PPO) algorithm (Schulman et al. (2017)). All experiments are conducted on a server equipped with eight NVIDIA L40s GPUs. Each policy is trained for 15,000 episodes with 4,096 parallel environments, typically requiring 13–15 hours to converge for a single motion primitive.

Language models: Across all components of our pipeline - task decomposition, skill selection, reward generation, and reward refinement, we evaluate six state-of-the-art LLMs *openai/gpt-4o*, *anthropic/claude-4.5-sonnet*, *google/gemini-2.5-flash*, *meta-llama/llama-3.3-70b-instruct*, *mistral-large*, *deepseek-v3*. These models were chosen for their strong performance in reasoning, code generation, and structured task synthesis. Each model is prompted identically to ensure a fair comparison across all experimental stages.

4.2 Skill selection

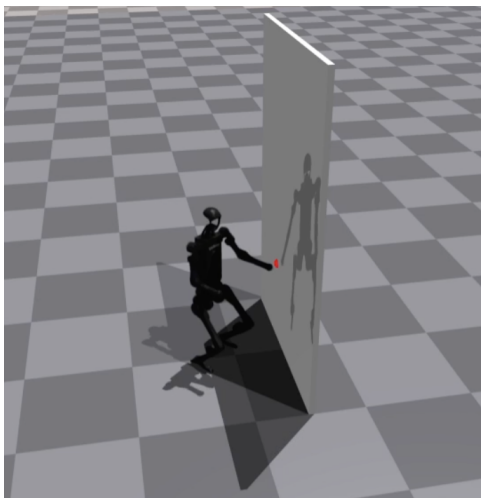
To assess whether language models can reliably decompose long-horizon tasks into the correct set of primitive skills, we evaluate the LLMs on the SkillBlender benchmark suite. Table 1 reports task-level correctness, where a model receives credit only if it predicts *all* required skills for a given task. The results reveal substantial variation across models and tasks.

GPT-4o achieves the highest accuracy (75%), consistently identifying core skills for locomotion-heavy tasks such as *FarReach*, *FootballShoot*, and *BoxTransfer*. However, even GPT-4o exhibits failures on tasks involving object manipulation or multi-stage interactions (e.g., *CabinetClose*, *PackageCarry*), suggesting that complex affordances remain challenging for purely language-based reasoning. Claude-4.5-Sonnet and Gemini-2.5-Flash perform moderately well (62.5%), but both models tend to miss secondary skills needed for contact-rich tasks. Notably, Gemini struggles on *FarReach*, indicating sensitivity to subtle distinctions between reaching and locomotion subtasks. Llama-3.3-70B and Mistral-Large achieve only

Model	Acc.	FR	BP	CC	FS	BX	PL	BT	PC
openai/gpt-4o	75.0%	✓	✓	✗	✓	✓	✓	✓	✗
anthropic/claude-4.5-sonnet	62.5%	✓	✓	✗	✗	✓	✓	✓	✗
google/gemini-2.5-flash	62.5%	✗	✓	✗	✓	✓	✓	✓	✗
meta-llama/llama-3.3-70b	50.0%	✓	✗	✗	✗	✓	✓	✓	✗
mistral-large	50.0%	✗	✓	✗	✓	✗	✓	✓	✗
deepseek-v3	37.5%	✓	✗	✗	✗	✓	✗	✓	✗

Table 1: Skill-selection accuracy across eight long-horizon humanoid tasks. Abbreviations: FR = FarReach, BP = ButtonPress, CC = CabinetClose, FS = FootballShoot, BX = BoxPush, PL = PackageLift, BT = BoxTransfer, PC = PackageCarry. Each cell indicates whether the model predicted the complete skill set for a task (✓) or missed at least one component (✗). Full task descriptions are provided in Appendix A.2.

(a) Squatting primitive



(b) Button press task

Figure 2: Representative motion tasks used for evaluating reward functions generated by Claude-4.5-Sonnet: (a) squatting (existing primitive skill) and (b) button press (composite high-level task).

50%, often omitting necessary stabilization or approach skills, reflecting weaker grounding in physical task structure. DeepSeek-V3 exhibits the lowest accuracy (37.5%) and frequently underpredicts the required skill set.

Overall, these findings highlight that while modern LLMs can reliably detect primary motion primitives, they still struggle with composite skills that require implicit reasoning about contact dynamics, sequencing, and dependencies between subtasks. This motivates our framework's downstream reward-generation and refinement pipeline, which aims to compensate for imperfect skill selection by enabling automatic acquisition of missing capabilities.

4.3 Reward generation for existing skills

In this section, we evaluate the reward generation capability of our framework. To simplify the process, we select two existing environments from SkillBlender: one primitive skill, squatting, and one composite skill, button press. We provide each LLM with the environment description file and configuration file, which specify all available reward functions, and prompt them to generate the corresponding reward values for each task. The humanoid robot is then trained using the LLM-generated rewards. After training convergence, we evaluate the performance of the resulting policies and compare them with the human-tuned baseline reward values provided by SkillBlender.

Model	Squatting (✓/✗)	Err. for squatting height (m)	Button (✓/✗)
openai/gpt-4o	✗	0.060	✗
anthropic/claude-4.5-sonnet	✓	0.051	✓
google/gemini-2.5-flash	✗	0.158	✗
meta-llama/llama-3.3-70b	✗	0.21	✗
mistral-large	✓	0.051	✗
deepseek-v3	✓	0.045	✗
Baseline (Ground truth)	✓	0.045	✓

Table 2: Reward generation results on two existing skills: *squatting* and *button press*. Each task is evaluated by success status, and the squatting task additionally reports the error of the squatting height.

4.3.1 Primitive skill: Squatting

The squatting primitive, illustrated in Fig. 2(a), requires the robot to lower its torso to a target height while maintaining balance and joint coordination. This task provides a clean benchmark for evaluating whether an LLM can generate stable and numerically well-shaped reward values for an already-existing primitive skill.

Table 2 summarizes the results. Claude, Mistral-Large, and DeepSeek-V3 successfully produce reward values that enable the robot to complete the squatting motion, achieving performance close to the ground-truth baseline. In contrast, GPT-4o, Gemini-2.5-Flash, and LLaMA-3.3-70B fail to generate effective reward values, with policies converging to either minimal movement or unstable oscillatory behaviors.

A key observation is that successful models tend to produce reward scalings numerically similar to the human-engineered values. Models that deviated significantly in the weighting of joint position tracking or base height shaping often failed to induce the intended lowering motion. These results highlight that while LLMs can reproduce structured reward templates, they remain sensitive to numerical misspecification, even for relatively simple low-dimensional skills like squatting.

4.3.2 Combined skill: Button press task

We further evaluate our framework's reward generation capability on a combined skill. Specifically, we select the button press task, illustrated in Fig. 2(b), in which a button is mounted on a wall at a random distance from the robot. The robot must first move toward the wall and then use its end-effector to press the button. Walking and reaching are the two primitive skills selected to finish this task.

The results in Table 2 indicate that only the reward values generated by Claude successfully enable the robot to complete the button press task. In contrast, all other models fail, with the corresponding policies causing the robot to simply stand still without approaching the target. This suggests that combined skills such as button pressing are difficult for LLMs to handle, even when all relevant reward functions are provided. These results are based on a single round of reward generation and iterative refinement may improve performance, which we leave for future work.

4.4 Reward generation for new skills

To evaluate our framework's capability to generate novel skills, we select eight representative behaviors, sway, punch, lie, jump, bend, wave, kick, and side-step, and prompt the LLM to synthesize corresponding reward functions. Detailed prompt instructions are provided in Appendix A.3.2. Figure 3 shows the screenshots from the simulation for the top performing model - GPT-4o.

The results show that among the eight evaluated skills, only side-step and bend were successfully learned. Kick and jump are considered partially successful: in the kick task, the robot frequently loses balance, likely due to the narrow foot structure of the Unitree H1. This suggests that the corresponding reward function may require more precise tuning to

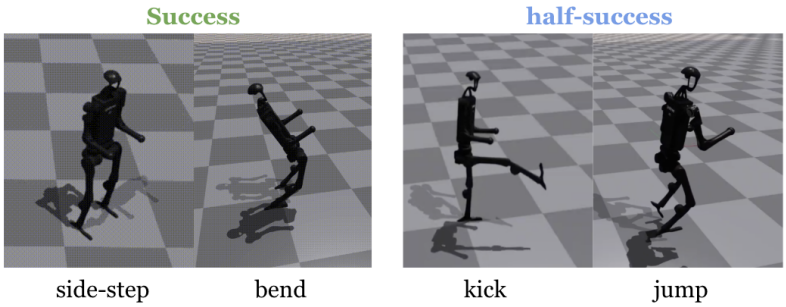


Figure 3: Screenshots from the Isaac Gym simulation for newly learned skills. The two figures on the left illustrate the successfully learned skills, *side-step* and *bend*, while the two figures on the right depict the partially successful skills, *kick* and *jump*.

maintain stability, as the kicking motion inherently demands strong balance control. For the jump task, the robot exhibits a tendency to jump but achieves only a very short airtime. We hypothesize that effective jumping requires a two-stage process, first squatting down, then pushing off, which may be difficult for an LLM-generated reward function to capture and coordinate.

As for the other four unsuccessful tasks like sway, punch, lie, and wave, the reasons for failure vary. The robot accidentally learned the side-step motion when prompted to learn sway, which may have resulted from a misused or overly general reward function. For punch, the reward function generated by the LLM consistently failed to execute, likely due to improper use of acceleration-related terms. The lie skill was initially proposed because the robot frequently fell during training, effectively learning an uncontrolled falling motion. However, lying down smoothly is a complex, whole-body contact and multi-stage behavior that remains challenging to capture through an LLM-generated reward alone. Finally, although wave was expected to be simple, the trained policy merely kept the robot standing still, indicating that the generated reward function did not effectively drive the intended motion.

5 Conclusion and Future Work

This work presents a self-evolving meta-skill learning framework in which a humanoid robot can autonomously acquire new motor capabilities using a closed-loop integration of large language models and reinforcement learning. Our system decomposes high-level natural-language tasks, identifies the primitive skills required for execution, and automatically generates reward functions for newly discovered skills. These rewards are then evaluated and refined based solely on RL performance metrics, enabling iterative improvement without human supervision. Through this pipeline, we demonstrate that modern LLMs are capable of producing structured, environment-consistent reward functions and that these generated rewards can successfully train both existing and novel skills. The framework provides a foundation for scalable, open-ended skill acquisition, moving toward autonomous humanoid agents that continually expand and improve their behavioral repertoire. While the proposed system shows strong promise, several important directions remain. First, a deeper study of reward refinement convergence is needed to determine when and how the iterative LLM–RL loop stabilizes. Second, incorporating sparse visual observations or image sequences into the refinement process could allow the LLM to reason directly about motion quality rather than relying purely on aggregated metrics. Third, evaluating the full pipeline on long-horizon, multi-stage tasks would provide insight into its ability to compose and reuse learned skills at scale.

Author Contributions

Xinyuan Luo led the implementation of the Isaac Gym simulation environment, executed all RL training experiments, and analyzed policy performance. Ajay Vikram Periasami developed the LLM-driven task decomposition, reward synthesis, and refinement modules, and conducted the multi-model ablation studies. Both authors collaboratively formulated the research direction, integrated the system components, interpreted results, and co-wrote the report.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

Siddhant Bhambri, Amrita Bhattacharjee, Durgesh Kalwar, Lin Guan, Huan Liu, and Subbarao Kambhampati. Extracting heuristics from large language models for reward shaping in reinforcement learning, 2024. URL https://arxiv.org/abs/2405.15194.

Rishi Hazra, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. Revolve: Reward evolution with large language models using human feedback, 2025. URL https://arxiv.org/abs/2406.01309.

Siyuan Huang, Zhengkai Jiang, Hao Dong, Yu Qiao, Peng Gao, and Hongsheng Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model, 2023. URL https://arxiv.org/abs/2305.11176.

Yuxuan Kuang, Haoran Geng, Amine Elhafsi, Tan-Dzung Do, Pieter Abbeel, Jitendra Malik, Marco Pavone, and Yue Wang. Skillblender: Towards versatile humanoid whole-body loco-manipulation via skill blending, 2025. URL https://arxiv.org/abs/2506.09366.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models, 2023. URL https://arxiv.org/abs/2303.00001.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control, 2023. URL https://arxiv.org/abs/2209.07753.

Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. Delta: Decomposed efficient long-term robot task planning using large language models, 2025. URL https://arxiv.org/abs/2404.03275.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2024. URL https://arxiv.org/abs/2310.12931.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021. URL https://arxiv.org/abs/2108.10470.

Yun Qu, Yuhang Jiang, Boyuan Wang, Yixiu Mao, Cheems Wang, Chang Liu, and Xiangyang Ji. Latent reward: Llm-empowered credit assignment in episodic reinforcement learning, 2025. URL https://arxiv.org/abs/2412.11120.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models, 2022. URL https://arxiv.org/abs/2209.11302.

Shengjie Sun, Runze Liu, Jiafei Lyu, Jing-Wen Yang, Liangpeng Zhang, and Xiu Li. A large language model-driven reward design framework via dynamic feedback for reinforcement learning, 2024. URL https://arxiv.org/abs/2410.14660.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Reward shaping with language models for reinforcement learning, 2024. URL https://arxiv.org/abs/2309.11489.

A Appendix

A.1 Example Task Decomposition Output

To illustrate how the Task Planner decomposes a natural-language instruction into structured subtasks and skills, we present a representative example for the *button press* task from the SkillBlender benchmark.

Task Description

> *"Walk up to a wall and press the button at shoulder height."*

Given this instruction and the current skill library, the LLM produces the following structured decomposition. Each subtask includes a concise subgoal, the full set of required skills, the subset of skills already present in the library, and any newly identified skills that must be learned.


```
{
  "task": {
    "name": "buttonpress",
    "description": "walk up to a wall and press the button at shoulder height"
  },
  "subtasks": [
    {
      "subgoal": "approach the wall",
      "skills": ["Walking"],
      "known_skills": ["Walking"],
      "new_skills": []
    },
    {
      "subgoal": "position body to press button",
      "skills": ["Stepping", "Reaching"],
      "known_skills": ["Stepping", "Reaching"],
      "new_skills": []
    },
    {
      "subgoal": "press the button",
      "skills": ["Reaching", "Pushing"],
      "known_skills": ["Reaching"],
      "new_skills": ["Pushing"]
    }
  ]
}
```

Discussion This example highlights several important aspects of our decomposition framework. First, the LLM correctly identifies that the task can be broken down into three sequential, physically grounded subtasks: locomotion toward the wall, body positioning relative to the button, and contact-based interaction. Second, the planner explicitly distinguishes between reusable skills already present in the library (e.g., *Walking*, *Reaching*) and missing capabilities that require new skill acquisition. In this case, the *Pushing* skill is automatically detected as absent and is subsequently passed to the reward synthesis and refinement pipeline for learning.

By producing a structured, machine-readable decomposition, the Task Planner enables seamless integration between high-level language understanding and low-level reinforcement learning, while also supporting scalable skill reuse and autonomous expansion of the skill library.

A.2 Task Descriptions

The long-horizon evaluation uses eight tasks from the SkillBlender benchmark, grouped by difficulty level.

Easy Tasks These tasks involve short-horizon interactions with minimal contact:

- **FarReach (FR)**: Reach two distant 3D points using both hands.
- **ButtonPress (BP)**: Press a wall-mounted button with the left wrist while keeping the right arm's pose.
- **CabinetClose (CC)**: Close an open cabinet in front of the humanoid.

Medium Tasks These tasks remain short-horizon but involve increased contact with objects and the environment:

- **FootballShoot (FS)**: Shoot a football toward a goal position.
- **BoxPush (BX)**: Push a box on a table to a target location.
- **PackageLift (PL)**: Lift a package to a specified height.

Hard Tasks These tasks involve rich contact dynamics and require multi-stage, long-horizon coordination:

- **BoxTransfer (BT)**: Transfer a box from one table to a target location on another table.
- **PackageCarry (PC)**: Carry a package to a distant location.

A.3 Prompts

Large Language Models play three roles in our framework: decomposing natural-language tasks into structured subtasks, generating reward functions for missing skills, and refining those rewards based on RL performance. We design specialized prompts for each stage to ensure consistency, interpretability, and reproducibility.

A.3.1 Task Decomposition

```
You are a high-level humanoid motion planner and meta-skill
developer.
Your task is to decompose a long-horizon natural-language goal into
a sequence of short, atomic subtasks. Each subtask must include a
concise ‘‘subgoal’’ (one line of natural language) and a list of
‘‘skills’’ required to achieve it.
You currently have access to the following SKILL LIBRARY:
```

{SKILL_LIBRARY}

For every subgoal, follow this reasoning process:

1. Compare the subgoal against the skills in the current library.
2. If ALL needed abilities can be performed using these existing skills, reuse them directly.
3. If the subgoal requires a new ability not in the list, invent a new skill:

- The new skill should be short (1--2 words), descriptive, and physically plausible for a humanoid (e.g., Grasping, Balancing, Turning, Lifting, Pushing).
- Avoid abstract or cognitive terms like ‘‘Planning’’ or ‘‘Deciding’’.
- Do not use synonyms of existing skills unless functionality differs.
- If none of the current skills fully cover a subgoal, you **must** create a new primitive skill name.

The final output **must** be a valid JSON object of this form:

```
{
  "subtasks": [
   {"subgoal": "<short actionable goal>", "skills": ["Skill1","Skill2", ...]},
    ...
  ]
}
```

Guidelines:

- Ensure subtasks are sequential and fully cover the task.
- Keep skill names capitalized and concise.
- Keep subgoals to one line.
- Return **only** the JSON with no explanations or comments.

A.3.2 Reward Generation

You are a code-generation agent designing a new humanoid skill reward function. Return only valid Python code defining the reward class. Do not include explanations, markdown fences, comments, or introductory text.

Below are the existing primitive skill rewards currently used:
<existing_rewards>
{reward_context}
</existing_rewards>

Analyze these to understand parameter conventions and scaling.

Below is the humanoid environment definition used for training:
<environment_context>
{env_context}
</environment_context>

Now, design a NEW reward class for the following subgoal and skill:

Subgoal: {subgoal}
Skill: {skill_name}

Follow these rules: 1. Maintain the same Python class structure (class rewards: with nested class scales:).
2. Reuse ONLY variables, parameters, and scale names already defined in existing reward functions or the environment context.
3. Do NOT invent new variables or physics metrics.

```
4.  Keep numerical scales realistic and consistent with the
reference rewards.
5. Return only valid Python code, with no explanations or comments.
```

A.3.3 Reward Refinement

```
You are a humanoid reward designer. The following reward function
was trained in RL but performed poorly.

Subgoal: {subgoal}
Skill: {skill}

[Generated reward]
{generated_reward}

[Observed RL feedback metrics]
{feedback_str}

Based on these feedback metrics, rewrite the reward function to:
1. improve stability, convergence, and success rate,
2.  keep structure consistent with existing reward format (class
rewards: with nested scales),
3. keep parameter names consistent, and
4.  adjust only scale magnitudes or add missing shaping terms if
necessary.

Output only the corrected Python code starting with class rewards:.
Do NOT include explanations or markdown.
```

A.4 Reinforcement learning reward curves

A.4.1 Existing skill reward curve

Figure 4: The left figure shows the reward curve during training for the existing primitive skill *squatting*, and the right figure shows the mean reward curve for the composite skill *button press*.

A.4.2 New skill reward curve

Figure 5: Mean reward curves for the newly generated skills: *sway*, *punch*, *lie*, *jump*, *bend*, *wave*, *kick*, and *side-step*.