CurricuLLM: Automatic Task Curricula Design for Learning Complex Robot Skills using Large Language Models

Kanghyun Ryu[1], Qiayuan Liao[1], Zhongyu Li[1], Payam Delgosha[2], Koushil Sreenath[1], Negar Mehr[1]

***Abstract*— Curriculum learning is a training mechanism in reinforcement learning (RL) that facilitates the achievement of complex policies by progressively increasing the task difficulty during training. However, designing effective curricula for a specific task often requires extensive domain knowledge and human intervention, which limits its applicability across various domains. Our core idea is that large language models (LLMs), with their extensive training on diverse language data and ability to encapsulate world knowledge, present significant potential for efficiently breaking down tasks and decomposing skills across various robotics environments. Additionally, the demonstrated success of LLMs in translating natural language into executable code for RL agents strengthens their role in generating task curricula. In this work, we propose CurricuLLM, which leverages the high-level planning and programming capabilities of LLMs for curriculum design, thereby enhancing the efficient learning of complex target tasks. CurricuLLM consists of: (Step 1) Generating a sequence of subtasks that aid target task learning in natural language form, (Step 2) Translating natural language description of subtasks in executable task code, including the reward code and goal distribution code, and (Step 3) Evaluating trained policies based on trajectory rollout and subtask description. We evaluate CurricuLLM in various robotics simulation environments, ranging from manipulation, navigation, and locomotion, to show that CurricuLLM can aid learning complex robot control tasks. In addition, we validate humanoid locomotion policy learned through CurricuLLM in the real-world. Project website is https://iconlab.negarmehr.com/CurricuLLM/**

I. INTRODUCTION

Deep reinforcement learning (RL) has achieved notable success across various robotics tasks, including manipulation [1], navigation [2], and locomotion [3]. However, RL requires informative samples for learning, and obtaining these from a random policy is highly sample-inefficient, especially for complex tasks. In contrast, human learning strategies differ significantly from random action trials; they typically start with simpler tasks and progressively increase difficulty. Curriculum learning, inspired by this structured approach of learning, aims to train models in a meaningful sequence [4], gradually enhancing the complexity of the training data [5] or the tasks themselves [6]. Particularly in RL, curriculum learning improves training efficiency by focusing on simpler tasks that can provide informative experiences to reach more complex target tasks, instead of starting from scratch [6].

*This work is supported by the National Science Foundation, under grants ECCS-2438314 CAREER Award, CNS-2423130, and CCF-2423131

[1]Mechanical Engineering, University of California Berkeley {kanghyun.ryu, qiayuanl, zhongyu_li, koushils, negar}@berkeley.edu

[2]This work was done when Payam was a research assistant professor at Computer Science department, University of Illinois at Urbana-Champaign. Payam is currently at Apple. delgosha@illinois.edu

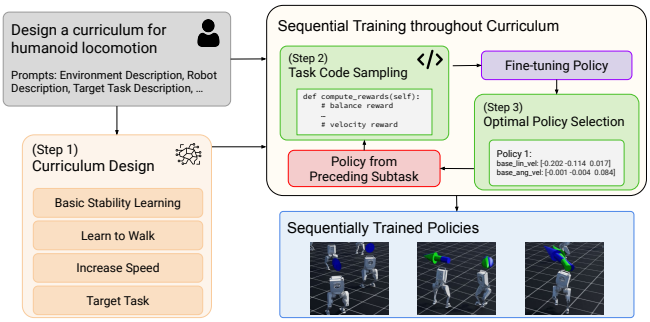


Fig. 1: CurricuLLM takes language descriptions of environments, robots, and the target task that we wish the robot to learn, and then generates a sequence of subtasks. In each subtask, CurricuLLM samples different task codes and evaluates the resulting trained policy to find the policy which is best aligned with the current subtask. These iterations are repeated throughout the curriculum subtasks to sequentially train a policy that reaches a complex target task.

Although effective, designing a good curriculum is challenging. Manual curriculum design often necessitates the costly intervention of human experts [7], [8], [9] and is typically restricted to a limited set of predefined tasks [10]. Consequently, several works focused on automatic curriculum learning (ACL). To generate task curricula, ACL requires the ability to determine subtasks aligned with the target task, rank the difficulty of each subtask, and organize them in the order of difficulty [11]. However, autonomously evaluating the relevance and difficulty of these subtasks remains unresolved. As a result, ACL has been limited to initial state curricula [12], [13], goal state curricula [14], or environment curricula [10], [15], rather than task-level curricula.

Meanwhile, in recent years, large language models (LLMs) trained on extensive collections of language data [16], [17], [18] have been recognized as repositories of world knowledge expressed in linguistic form [19]. Leveraging this world knowledge, LLMs have demonstrated their capabilities in task planning [20] and skill decomposition for complex robotic tasks [21], [22]. Furthermore, the programming skills of LLMs enabled smooth integration between high-level language description and robotics through API call composition [23], [24], simulation environment generation [25], [26], or reward design [27], [28].

In this paper, we introduce CurricuLLM, which leverages the reasoning and coding capabilities of LLMs to design curricula for complex robotic control tasks. Our goal is to autonomously generate a series of subtasks that facilitate the learning of complex target tasks without the need for extensive human intervention. Utilizing the LLM's task de-

composition and coding, CurricuLLM autonomously generates sequences of subtasks along with appropriate reward functions and goal distributions for each subtask, enhancing the efficiency of training complex robotic policies.

Our contribution can be summarized in threefold. First, we propose CurricuLLM, a task-level curriculum designer that uses the high-level planning and code writing capabilities of LLMs. Second, we evaluate CurricuLLM in diverse robotics simulation environments ranging from manipulation, navigation, and locomotion, demonstrating its efficacy in learning complex control tasks. Finally, we validate the policy trained with CurricuLLM on the Berkeley Humanoid [29], illustrating that the policy learned through CurricuLLM can be transferred to the real world.

II. Related Works

A. Curriculum Learning

In RL, curriculum learning is recognized for enhancing sample efficiency [30], addressing previously infeasible challenging tasks [31], and facilitating multitask policy learning [32]. Key elements of curriculum learning include the difficulty measure, which ranks the difficulty of each subtask, and training scheduling, which arranges subtasks at an appropriate pace [11]. The teacher-student framework, for example, has a teacher agent that monitors the progress of the student agent, recommending suitable tasks or demonstrations accordingly [33], [34]. However, this method requires a predefined set of tasks provided by human experts or a teacher who has superior knowledge of the environment. Although self-play has been proposed as a means to escalate opponent difficulty [35], [36], it is limited to competitive multi-agent settings and may converge to a local minimum.

An appropriate difficulty measure is also crucial in curriculum learning. In goal-conditioned environments, it is often suggested to start training from a goal distribution close to the initial state [14], [37] or an initial state distribution that is in proximity to the goal state [12], [13] to regulate difficulty. However, these methods are limited to goal-conditioned environments where the task difficulty is correlated with how "far" the goal is from the start location. In this work, we use LLM to provide a more general method to measure task difficulty and design curricula.

The most closely related work to CurricuLLM is DrEureka [38] and Eurekaverse [26], which utilizes LLMs to generate domain randomization parameters, such as gravity or mass, or environment curriculum, such as terrain height. Especially, Eurekaverse employs a co-evolution mechanism that gradually increases the complexity of the environment by using the LLM for environment code generation. Compared to Eurekaverse, our method focuses on the *task* curriculum, which focuses on task break-down for learning complex robotic tasks compared to the Eurekaverse which focuses on generalization across different environments.

B. Large Language Model for Robotics

Task Planning. The robotics community has recently been exploring the use of LLMs for high-level task planning [39], [24], [40]. However, these methods are limited to task planning within predefined finite skill sets and suffer when LLMs' plan is not executable within given skill sets or environment [20], [24], [41]. In contrast, Voyager [21] introduces an automatic skill discovery, attempting to learn new skills that are not currently available but required for open-ended exploration. Nonetheless, skills in Voyager are limited to composing discrete actions and are inapplicable to continuous control problems. In this work, we propose a task curriculum generation that facilitates the efficient learning of continuous control tasks. To manage the control of robots with high degrees of freedom, we utilize the coding capabilities of LLMs to generate a reward function for each subtask and sequentially train each subtask in the given order.

Reward Design. In continuation from works using natural language as a reward [42], [43], several works have proposed using LLMs as a tool to translate language to reward. For example, [44] uses LLMs to translate motion description to cost parameters, which are optimized using model predictive control (MPC). However, they are limited to changing the parameters in cost functions that are hand-coded by human experts. On the other hand, some works proposed directly using LLM [45] or vision-language model (VLM) [46] as a reward function, which observes agent behavior and outputs a reward signal. However, these approaches require expensive LLM or VLM interaction during training. Most similar approaches to our work are [27], [47], which leverage LLMs to generate reward functions and utilize an evolutionary search to identify the most effective reward function. However, these methods require an evaluation metric in their feedback loops, and their reward search tends to optimize specifically for this metric. Therefore, for tasks only described in natural language, as subtasks in our curriculum, finding reward functions without these evaluation metrics can be challenging. Additionally, their evolutionary search is highly sample-inefficient, contradicting the efficient learning objectives of curriculum learning. In contrast, our work divides a single complex task into a series of subtasks, and then employs a reasoning approach analogous to the chain of thoughts [19] to generate reward functions for complex target tasks.

III. Problem Formulation

In this work, we consider task curriculum generation for learning control policies for complex robot tasks. First, we model a (sub)task as a goal-conditioned Markov Decision Process (MDP), formally represented by a tuple $m = (\mathcal{S}, \mathcal{G}, \mathcal{A}, p, r, \rho_g)$. Here, $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $p(s'|s,a)$ is a transition probability function, $r(s,a,s',g)$ is a reward function, $\mathcal{G}$ is a goal space, and ρ_g is a goal distribution. We use subscript n to describe n_{th} subtask in our curriculum. Then, following [6], we formally define a task curriculum as:

Definition 1 (Task-level Sequence Curriculum. [6]): A task-level sequence curriculum can be represented as an ordered list of tasks $C = [m_1, m_2, \ldots, m_N]$ where if $i \leq j$ for some $m_i, m_j \in C$, then the task m_i should be learned before task m_j.

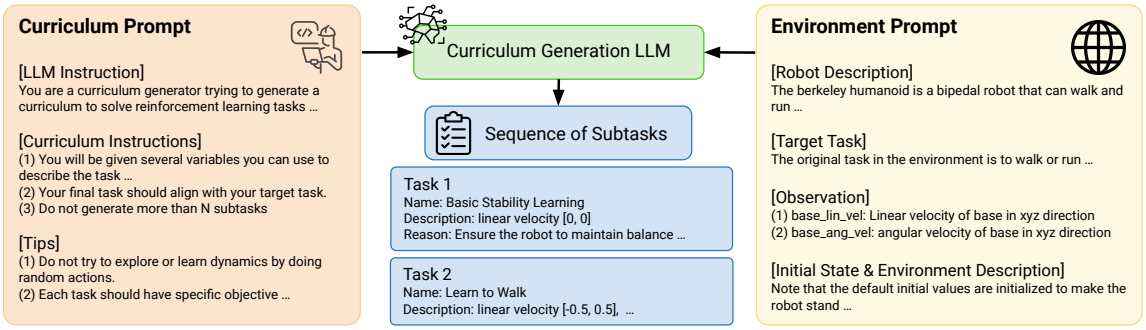


Fig. 2: (Step 1) Curriculum generation LLM receives the natural language form of a curriculum prompt as well as the environment description to generate a sequence of subtasks. Our prompt includes instructions for the curriculum designer, rules for how to describe the subtasks, and other tips on describing the curriculum. Environment description consists of the robot and its state variable description, the target task, and the initial state description.

In our work, we aim to generate a task curriculum $C = [m_1, m_2, \ldots, m_N]$ which helps learning a policy π that maximizes the cumulative reward $V_{m_T}(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1}, g)\right]$ for the target task m_T.

Problem 1: For target task $m_T = (\mathcal{S}, \mathcal{G}, \mathcal{A}, p, r_T, \rho_{g,T})$ and task curriculum $C = [m_1, m_2, \ldots, m_N]$, we denote a policy trained with curriculum C as π_C. Our objective for curriculum design is finding a curriculum C that maximizes the reward on the target task $\arg\max_C V_{m_T}(\pi_C)$.

We assume the state space $\mathcal{S}$, the action space $\mathcal{A}$, the transition probability function p, and the goal space $\mathcal{G}$ are fixed, i.e., they do not change between subtasks. Therefore, we can specify the task curriculum with the sequence of reward functions and goal distributions that are associated with the subtasks, $C_{(r,\rho_g)} = [(r_1, \rho_{g,1}), (r_2, \rho_{g,2}), \ldots, (r_N, \rho_{g,N})]$. Here, we express the reward function and goal distribution tuple (r, ρ_g) with a programming code for the simulation environment and define it as a *task code*. Therefore, CurricuLLM's objective reduces to generating a sequence of task codes $C_{(r,\rho_g)}$ that maximize the target task performance.

IV. Method

Even for LLMs encapsulating world knowledge, directly generating the sequence of task codes $C_{(r,\rho_g)}$ can be challenging. Since LLM is known to show better reasoning capability by following step-by-step instructions [19], we divide our curriculum generation into three main modules (see Figure 1):

- Curriculum Design (Step 1): A curriculum generation LLM receives natural language descriptions of the robot, environment, and target task to generate sequences of language descriptions $C_l = [l_1, l_2, \ldots, l_N]$ of the task sequence curriculum C.
- Task Code Sampling (Step 2): A task code generation LLM generates K task code candidates $(r_n^k, \rho_{g,n}^k)$, $k = \{1, 2, \ldots, K\}$ for the given subtask description l_n. These are in the form of executable code and are used to fine-tune the policy trained for the previous subtask.
- Optimal Policy Selection (Step 3): An evaluation LLM evaluates policies π_n^k, $k = \{1, 2, \ldots, K\}$ trained with different task code candidates $(r_n^k, \rho_{g,n}^k)$, $k = \{1, 2, \ldots, K\}$ to identify the policy that best aligns with the current subtask. Selected policy π_n^* is used as a pretrained policy for the next subtask.

A. Generating Sequence of Language Description

Leveraging the high-level task planning from LLMs, we initially ask an LLM to generate a series of language descriptions $C_l = [l_1, l_2, \ldots, l_N]$ for a task-level sequence curriculum $C = [m_1, m_2, \ldots, m_N]$ facilitating the learning of a target task m_T. Initially, the LLM is provided with a language description of the target task l_T and a language description of the environmental information l_E to generate an environment-specific curriculum. When generating curriculum, we query LLM to use the target task m_T as a final task in curriculum m_N. Moreover, we require the LLM to describe the subtask using available state variables. This enables the LLM to generate curricula that are grounded in the environment information and ensures the generation of a reliable reward function later (discussed in Section IV-B). For example, to generate a curriculum for a humanoid to learn running, we query the LLM to generate a curriculum for following a high speed commands consist of linear velocity and heading angle, while providing state variables, such as *base linear velocity*, *base angular velocity*, or *joint angle*. Then, the LLM generates the sequence of subtasks descriptions such as (1) Basic Stability Learning: Maintain stability by minimizing the joint deviation and height deviation, (2) Learn to Walk: Follow low speed commands, and others.

B. Task Code Generation

After generating a curriculum with a series of language descriptions $C_l = [l_1, l_2, \ldots, l_N]$, we should translate these language descriptions to a sequence of task codes $C_{(r,\rho_g)} = [(r_1, \rho_{g,1}), (r_2, \rho_{g,2}), \ldots, (r_N, \rho_{g,N})]$. These task codes are described in executable code so that the RL policy can be trained on these subtasks. In the n_{th} subtask which we denote by m_n, the task code generation LLM samples K different task codes $(r_n^k, \rho_{g,n}^k)$, $k \in \{1, 2, \ldots, K\}$ using the language description of the subtask l_n. Then, we obtain policy candidates π_n^k by fine-tuning the policy trained from the previous subtask π_{n-1}^* with task code $(r_n^k, \rho_{g,n}^k)$.

While previous works focused on evaluation metrics and feedback [27], [47] for evaluating policies with an LLM, we propose several strategies to identify the appropriate task

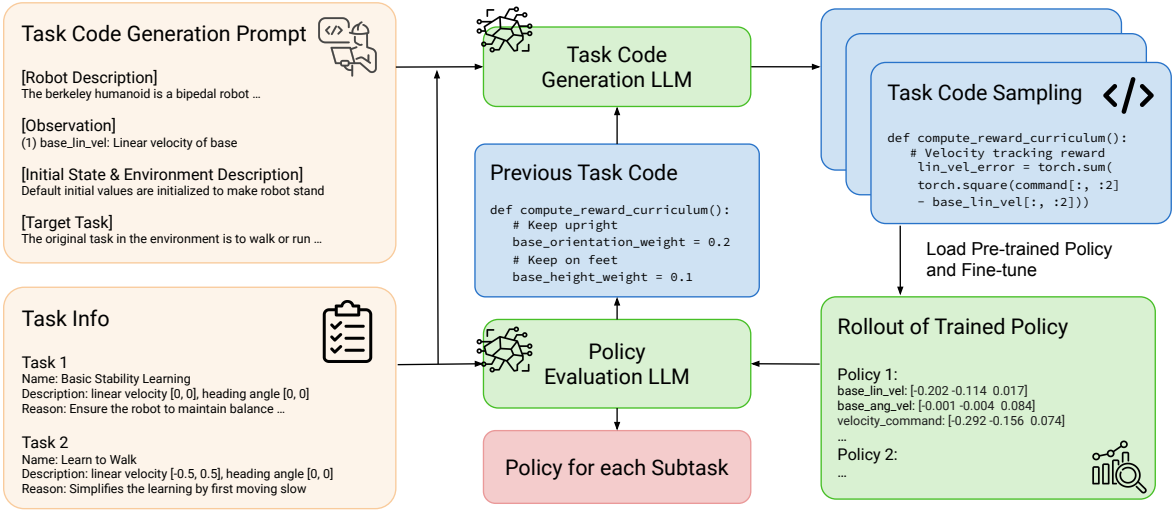


Fig. 3: Our task code generation and evaluation framework in each subtask. (Step 2) Task code generation LLM takes the environment and target task description, current and past task information, and the reward function used for previous tasks. Then, K task code candidates for the current subtask are sampled and used to fine-tune policies from the previous subtask. (Step 3) Then, evaluation LLM receives the statistics of trajectory rollout from trained policy and finds a policy that best aligns with the current subtask description.

code only with language description and without sample inefficient iterative searches. First, as proposed in IV-A, our subtask descriptions are grounded in state variables. This approach minimizes ambiguity when translating natural language into task code. Moreover, we provide normalized state variables for defining the reward functions instead of raw state values. This enables the LLM to find proper weighting parameters in reward function generation without iterative trial-and-error. In our experiments, no more than 5 samples are required for capturing feasible task code among potential code errors or variations in weight choices.

It is crucial for the generated reward function to not only facilitate learning of the current subtask but also to *remember* and utilize *previously* learned subtasks. To achieve this, we record the history of trained subtasks and their corresponding task codes, denoted as h, and provide this information to the LLM when generating the task code for the next subtask. This approach is analogous to the Chain-of-Thoughts prompting technique [19], enabling the generation of task code for complex subtasks in the later stages by building upon the simpler reward functions that were developed for earlier subtasks. Moreover, we instruct the LLM on how to adjust the weights of reward arguments: placing a greater emphasis on rewards related to the current task while still maintaining relevance to previous tasks to mitigate forgetting. This *chain of task code* ensures a comprehensive and adaptive learning process across the curriculum.

C. Large Language Model for Policy Evaluation

After training K different candidate policies $\pi_n^k, k \in \{1, \ldots, K\}$ using K different task codes $(r_n^k, \rho_{g,n}^k), k \in \{1, \ldots, K\}$, we evaluate the candidate polices and select the optimal policy π_n^* that best aligns with current subtasks. While previous approaches relied on human feedback [48] or predefined evaluation metrics [27], [47] to assess training outcomes, we evaluate the policy only with the language description of the current subtask.

Algorithm 1 Overall algorithm of CurricuLLM

Input Language description of target task l_T, Language description of environment l_E

Hyperparameter Number of task code samples K, Total number of training timesteps for each subtask M

Generate curriculum from LLM
$[l_1, l_2, \ldots, l_N] \sim LLM_{Curriculum}(l_T, l_E)$
$\pi \sim$ random policy # Initialize with random policy
$h = [\,]$
for l_n for $n = 1 : N$ **do**
  for $k = 1 : K$ **do**
    # Sample task code from LLM
    $(r_n^k, \rho_{g,n}^k) \sim LLM_{TaskCodeGeneration}(l_n, l_E, h)$
    $\pi_n^k \sim \text{train}(\pi, r_n^k, \rho_{g,n}^k, M)$
  end for
  # Roll out trajectories from policy candidates
  $\tau_n^1, \tau_n^2, \ldots, \tau_n^K \sim \pi_n^1, \pi_n^2, \ldots, \pi_n^K$
  # Evaluate the resulting trajectories
  $(r_n^*, \rho_{g,n}^*), \pi_n^* \sim LLM_{Evaluation}(l_n, \tau_n^1, \tau_n^2, \ldots, \tau_n^K)$
  # Update the policy and training history
  $\pi \leftarrow \pi_n^*$
  $h.append(l_n, r_n^*, \rho_{g,n}^*)$
end for

Output Policy π

To achieve this, we focused on the trajectory analysis capabilities of LLMs shown in [49]. We provide the LLM with the statistics of trajectory rollout $\tau_n^1, \tau_n^2, \ldots, \tau_n^K$ from each trained policy $\pi_n^1, \pi_n^2, \ldots, \pi_n^K$. Similar to the task code generation process, task history information h is given to the evaluation LLM. The LLM is then instructed to find the most effective task code for the current subtask description l_n, while also considering previous subtasks to avoid forgetting. This evaluation feedback mitigates randomness in LLM generated task code while ensuring learning with historical awareness.

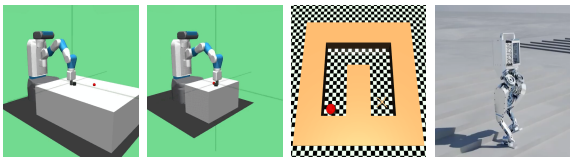
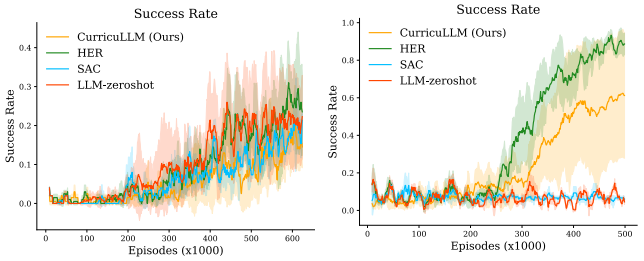
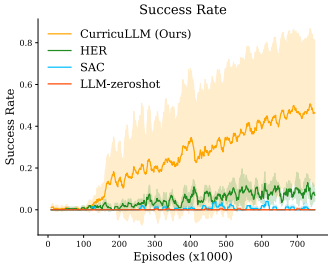


Fig. 4: Snapshot of Environments: From left to right, Fetch-Slide, Fetch-Push, AntMaze-UMaze, and Berkeley Humanoid.



(a) Fetch-Slide environment (b) Fetch-Push environment



(c) AntMaze-UMaze environment

Fig. 5: Success rate of tasks in Gymnasium-Robotics environments.

Finally, once the best policy π_n^* for the current subtask m_n is identified, this policy is archived in the library and utilized as a pre-trained policy for the subsequent subtask. Additionally, we update the history h with the selected task code to keep track of the trained subtasks. Our overall algorithm is summarized in Algorithm 1.

V. Experiments

In this Section, we evaluate CurricuLLM across different robotics tasks, ranging from manipulation, navigation, and locomotion, covering a broad range of robotic tasks. We first show CurricuLLM's efficiency in manipulation and navigation tasks in Section V-A. Then, we evaluate CurricuLLM in a higher dimension humanoid control problem in Section V-B. Finally, we deploy the policy learned through CurricuLLM to Berkeley Humanoid [29] to validate it in hardware.

A. Gymnasium Environments

For the manipulation and navigation tasks, we evaluate CurricuLLM in Fetch-Slide, Fetch-Push, and AntMaze-UMaze environments in Gymnasium-Robotics [50] with Stable-Baselines3 [51]. In the Fetch-Slide and Fetch-Push tasks, the 7DOF Fetch robot tries to manipulate a block to a desired goal position either by sliding or pushing the block. In the AntMaze environment, the ant robot tries to navigate the U-shaped maze and reach the goal position. While CurricuLLM was not able to change goal distributions in Fetch environments, in AntMaze, CurricuLLM was able to change the maximum distance between the initial ant

CurricuLLM generated reward for AntMaze

```
def compute_reward_curriculum(self):
    # Calculate the magnitudes
    velocity_magnitude = np.linalg.norm(torso_velocity)
    angular_velocity_magnitude =
        np.linalg.norm(torso_angular_velocity)
    # As goal_distance is received as an array but
    expected to be treated as scalar
    goal_distance_magnitude
        = np.linalg.norm(goal_distance)
    # Weighting parameters setup reflecting curriculum learning
    velocity_weight = 0.15
    # Substantial reduction to focus on goal achievement
    angular_velocity_weight = 0.15
    # Maintain orientation control importance
    goal_distance_weight = 0.5
    # Continuing to incentivize movement towards goal,
    but with lesser intensity due to the new success condition
    ...
```

LLM-zeroshot generated reward for AntMaze

```
def compute_reward_curriculum(self):
   # Define reward for reaching the goal
   success_reward_weight = 10.0
   success_reward = 0.0
   if goal_distance < 0.45:
       success_reward = 1.0

   # Calculate total reward
   reward = success_reward_weight * success_reward
```

Fig. 6: Example reward codes generated from CurricuLLM and reward code generated in a zero-shot manner from LLM for AntMaze environment. The reward code designed by CurricuLLM encompasses diverse behaviors that motivate learning the target task while directly querying a reward code for the target task leads to a sparse-reward, uninformative reward function.

position and the goal position in each subtask. We compare our method with three baselines: soft actor-critic (SAC) [52], hindsight experienced replay (HER) [53], and LLM-zeroshot. LLM-zeroshot baseline generates reward functions for the target task in a zero-shot manner from the LLM. Comparison with LLM-zeroshot highlights that CurricuLLM is necessary for learning complex control tasks and generating high-quality reward functions from LLM. While CurricuLLM is compatible with any RL algorithm, we implement it with SAC to have a fair comparison with HER, which has to use an off-policy algorithm. Each algorithm was trained with 5 different random seeds, while LLM-zeroshot was trained with 10 different seeds to reflect the randomness of LLM output. Finally, we use OpenAI's GPT-4-turbo [16] as LLM agents throughout the experiments.

Results. The learning curves of the success rates for each target task are illustrated in Figure 5. In every task, especially in the Fetch-Push and AntMaze-UMaze, CurricuLLM exhibited a comparable or superior success rate compared to the baselines. It is noteworthy that CurricuLLM outperformed every baseline in the most challenging AntMaze environment, which highlights the significance of curriculum in complex tasks. Furthermore, while HER is only applicable to goal-conditioned policy due to their relabeling scheme, CurricuLLM achieved comparable results in Fetch environments by only changing the reward function throughout the curriculum. Finally, we mention that the success rate of CurricuLLM was highly dependent on curriculum quality, resulting in a high variance in success rates.

Quality of Reward Code. In Figure 6, we compare the final reward code generated by CurricuLLM with the reward code derived from LLM-zeroshot in the AntMaze environ-

TABLE I: Results of Berkeley Humanoid Simulation

	Tracking Error (m/s)	Failure (%)	Episode Length (steps)
CurricuLLM	$\mathbf{0.46 \pm 0.38}$	$\mathbf{8.79 \pm 4.72}$	315.44 ± 73.95
LLM-zeroshot	1.13 ± 0.57	28.57 ± 45.19	146.66 ± 114.12
Human reward	$\mathbf{0.41 \pm 0.10}$	$\mathbf{7.31 \pm 3.12}$	397.67 ± 42.04
Human reward with live bonus	1.37 ± 0.66	0.51 ± 0.23	167.57 ± 98.22

ment. We observe that the reward code from CurricuLLM captures diverse behaviors from subtasks that are beneficial for achieving the target task, such as incentivizing fast movement for exploration and regulating angular velocity for stabilization. In contrast, the reward code from LLM-zeroshot focuses solely on the target task without such terms that facilitate learning. This demonstrates that our two-step approach—(1) generating a task-level curriculum and (2) evolving the task code effectively facilitates the generation of an informative reward function.

B. Berkeley Humanoid

1) Simulation Results: To show the effectiveness of CurricuLLM in the higher dimension, complex locomotion task, we use Berkeley Humanoid [29] environment implemented in Isaac Lab [54]. Berkeley Humanoid is a mid-size humanoid robot consisting of a state dimension of 45 and an action dimension of 12, which is desired joint positions. The target task is making the robot run by following a random command within $\{(v_x, v_y, \theta) | -2 \leq v_x, v_y \leq 2, -\pi \leq \theta \leq \pi\}$, where v_x, v_y are linear velocity in x and y direction, and θ is the heading angle. In curricula design, CurricuLLM was capable of specifying the command range and reward function for each subtask. We compare with the policies trained with two human-designed reward functions, the one proposed in the original paper and the other adding alive bonus to such reward. We also compare with LLM-zeroshot method. We excluded HER since it is only applicable to goal-conditioned sparse reward environments. All algorithms are implemented with proximal policy optimization (PPO) [55].

To ensure safe and desirable motion in the real-world, we add pre-defined reward functions in addition to the LLM generated rewards, such as removing high-frequency actions or avoiding sliding motion. In addition, we add a termination condition based on the accumulated tracking error following [7]. This termination condition is necessary for prohibiting the robot from achieving suboptimal behavior, such as sacrificing command tracking for stability.

Results. Simulation results are presented in Table I. An episode is defined as failed if the robot base contacted ground before 2 seconds. CurricuLLM achieved comparable results with the complex human-designed reward in terms of tracking error and failure rate. Considering that human reward is designed after tedious trial-and-error from robot engineers, it is significant that CurricuLLM achieved comparable results without any previous knowledge. LLM-zeroshot method tends to converge either to an unstable policy that terminates in the early stages or to a policy that sacrifices tracking error for stability. This is reflected in the high standard deviations of each performance metric. Interestingly, human reward with alive bonus converged into a behavior that ignores high speed command. We suspect adding an alive bonus made the policy prioritize learning a stable policy over risking instability from high speed movements. This illustrates that reward tuning is highly sensitive and requires extensive trial-and-error even for human experts, which motivates the automatic reward generation and training framework as CurricuLLM.

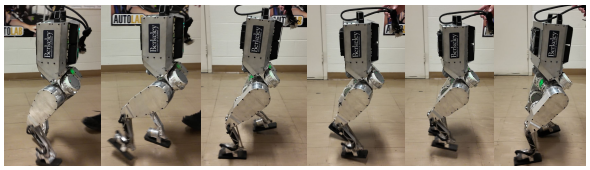


Fig. 7: Snapshot of hardware experiment.

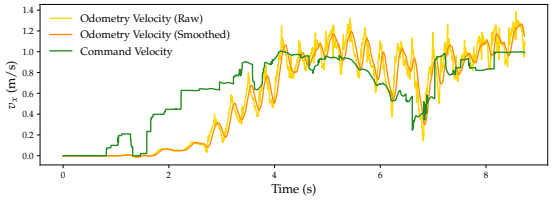


Fig. 8: Odometry data and command data of base linear velocity v_x in hardware experiments. The policy trained with CurricuLLM successfully follows the command in real-world.

2) Hardware Validation: In this section, we validate the locomotion policy learned through CurricuLLM on the Berkeley Humanoid. The command tracking performance in the real-world experiment is given in Figure 8. In the hardware experiments, we had to restrict the demonstration to walking within the command velocity $\{(v_x, v_y, \theta) | -1 \leq v_x, v_y \leq 1, -\pi \leq \theta \leq \pi\}$ due to the restrictions that are specific to the hardware and not related to the control policy. Through our hardware experiments, we found that the base velocity of the humanoid was able to accurately track the given command. This highlighted the effectiveness of the control policy learned through CurricuLLM in controlling the robot in the real world.

VI. CONCLUSIONS

In this paper, we introduced CurricuLLM, an automated task curriculum generator using LLMs. CurricuLLM first generates a language description of a sequence of subtasks that facilitate the learning of a complex target task. In each subtask, the task code generation LLM translates the subtask descriptions into task codes, which include a reward function and a goal distribution. After training policies for the sampled task codes, the evaluation LLM analyzes the trajectory of each policy and selects the best aligned policy for the subtask description. This optimal policy is utilized as a pre-trained model for subsequent subtasks. We showed that CurricuLLM successfully created task curricula and learned complex robotic control tasks in manipulation, navigation, and locomotion. We plan to extend our feedback loop by integrating training progress, such as learning curves, to optimize the training procedure by enabling adaptive training such as early stopping.

REFERENCES

[1] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396, IEEE, 2017.

[2] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364, IEEE, 2017.

[3] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement learning for robust parameterized locomotion control of bipedal robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2811–2817, IEEE, 2021.

[4] P. Soviany, R. T. Ionescu, P. Rota, and N. Sebe, "Curriculum learning: A survey," *International Journal of Computer Vision*, vol. 130, no. 6, pp. 1526–1565, 2022.

[5] A. Pentina, V. Sharmanska, and C. H. Lampert, "Curriculum learning of multiple tasks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5492–5500, 2015.

[6] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *Journal of Machine Learning Research*, vol. 21, no. 181, pp. 1–50, 2020.

[7] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement learning for versatile, dynamic, and robust bipedal locomotion control," *arXiv preprint arXiv:2401.16889*, 2024.

[8] Z. Tang, D. Kim, and S. Ha, "Learning agile motor skills on quadrupedal robots using curriculum learning," in *International Conference on Robot Intelligence Technology and Applications*, vol. 3, 2021.

[9] B. Tidd, N. Hudson, and A. Cosgun, "Guided curriculum learning for walking over complex terrain," in *Australasian Conference on Robotics and Automation 2020*, Australian Robotics and Automation Association (ARAA), 2020.

[10] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*, pp. 91–100, PMLR, 2022.

[11] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 9, pp. 4555–4576, 2021.

[12] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *Conference on robot learning*, pp. 482–495, PMLR, 2017.

[13] B. Ivanovic, J. Harrison, A. Sharma, M. Chen, and M. Pavone, "Barc: Backward reachability curriculum for robotic reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 15–21, IEEE, 2019.

[14] S. Forestier, R. Portelas, Y. Mollard, and P.-Y. Oudeyer, "Intrinsically motivated goal exploration processes with automatic curriculum learning," *Journal of Machine Learning Research*, vol. 23, no. 152, pp. 1–41, 2022.

[15] H.-C. Wang, S.-C. Huang, P.-J. Huang, K.-L. Wang, Y.-C. Teng, Y.-T. Ko, D. Jeon, and I.-C. Wu, "Curriculum reinforcement learning from avoiding collisions to navigating among movable obstacles in diverse environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2740–2747, 2023.

[16] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[17] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[18] M. Reid, N. Savinov, D. Teplyashin, D. Lepikhin, T. Lillicrap, J.-b. Alayrac, R. Soricut, A. Lazaridou, O. Firat, J. Schrittwieser, *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[19] 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. 24824–24837, 2022.

[20] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on robot learning*, pp. 287–318, PMLR, 2023.

[21] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," *arXiv preprint arXiv:2305.16291*, 2023.

[22] 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," in *The Twelfth International Conference on Learning Representations*, 2024.

[23] 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)*, pp. 9493–9500, IEEE, 2023.

[24] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530, IEEE, 2023.

[25] 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," in *Forty-first International Conference on Machine Learning*, 2024.

[26] W. Liang, S. Wang, H.-J. Wang, Y. J. Ma, O. Bastani, and D. Jayaraman, "Environment curriculum generation via large language models," in *8th Annual Conference on Robot Learning*, 2024.

[27] 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," in *The Twelfth International Conference on Learning Representations*, 2024.

[28] H. Li, X. Yang, Z. Wang, X. Zhu, J. Zhou, Y. Qiao, X. Wang, H. Li, L. Lu, and J. Dai, "Auto mc-reward: Automated dense reward design with large language models for minecraft," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16426–16435, 2024.

[29] Q. Liao, B. Zhang, X. Huang, X. Huang, Z. Li, and K. Sreenath, "Berkeley humanoid: A research platform for learning-based control," *arXiv preprint arXiv:2407.21781*, 2024.

[30] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[31] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer, "Automatic curriculum learning for deep rl: a short survey," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pp. 4819–4825, 2021.

[32] A. Jabri, K. Hsu, A. Gupta, B. Eysenbach, S. Levine, and C. Finn, "Unsupervised curricula for visual meta-reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[33] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher–student curriculum learning," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3732–3740, 2019.

[34] E. Clark, K. Ryu, and N. Mehr, "Adaptive teaching in heterogeneous agents: Balancing surprise in sparse reward scenarios," in *Proceedings of the 6th Annual Learning for Dynamics & Control Conference*, vol. 242 of *Proceedings of Machine Learning Research*, pp. 1489–1501, PMLR, 15–17 Jul 2024.

[35] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[36] Y. Du, P. Abbeel, and A. Grover, "It takes four to tango: Multiagent self play for automatic curriculum generation," in *International Conference on Learning Representations*, 2022.

[37] S. Lee, D. Cho, J. Park, and H. J. Kim, "Cqm: Curriculum reinforcement learning with a quantized world model," *Advances in Neural Information Processing Systems*, vol. 36, pp. 78824–78845, 2023.

[38] Y. J. Ma, W. Liang, H.-J. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman, "Dreureka: Language model guided sim-to-real transfer," *arXiv preprint arXiv:2406.01967*, 2024.

[39] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International conference on machine learning*, pp. 9118–9147, PMLR, 2022.

[40] Y. Ouyang, J. Li, Y. Li, Z. Li, C. Yu, K. Sreenath, and Y. Wu, "Long-horizon locomotion and manipulation on a quadrupedal robot with large language models," *arXiv preprint arXiv:2404.05291*, 2024.

[41] D. Shah, A. T. Toshev, S. Levine, and brian ichter, "Value function spaces: Skill-centric state abstractions for long-horizon reasoning," in *International Conference on Learning Representations*, 2022.
[42] S. Mirchandani, S. Karamcheti, and D. Sadigh, "Ella: Exploration through learned language abstraction," *Advances in neural information processing systems*, vol. 34, pp. 29529–29540, 2021.
[43] P. Goyal, S. Niekum, and R. Mooney, "Using natural language for reward shaping in reinforcement learning," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019.
[44] 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," in *Conference on Robot Learning*, pp. 374–404, PMLR, 2023.
[45] M. Kwon and S. Michael, "Reward design with language models," in *International Conference on Learning Representations (ICLR)*, 2023.
[46] J. Rocamonde, V. Montesinos, E. Nava, E. Perez, and D. Lindner, "Vision-language models are zero-shot reward models for reinforcement learning," in *The Twelfth International Conference on Learning Representations*, 2024.
[47] J. Song, Z. Zhou, J. Liu, C. Fang, Z. Shu, and L. Ma, "Self-refined large language model as automated reward function designer for deep reinforcement learning in robotics," *arXiv preprint arXiv:2309.06687*, 2023.
[48] J. Liang, F. Xia, W. Yu, A. Zeng, M. G. Arenas, M. Attarian, M. Bauzá, M. Bennice, A. Bewley, A. Dostmohamed, *et al.*, "Learning to learn faster from human feedback with language model predictive control," *CoRR*, 2024.
[49] Y.-J. Wang, B. Zhang, J. Chen, and K. Sreenath, "Prompt a robot to walk with large language models," *arXiv preprint arXiv:2309.09969*, 2023.
[50] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry, "Gymnasium robotics," 2023.
[51] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.
[52] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.
[53] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.
[54] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023.
[55] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.