ACL-LLM: Automatic Curriculum Learning via Large Language Models

Hansung Kim
hansung@berkeley.edu
SID: 3037384309

Yuxin Miao
yuxin_miao@berkeley.edu
SID: 3038587398

Code: https://github.com/hansungkim98122/285_project

***Index Terms*—curriculum learning, large language models, reinforcement learning**

I. Introduction

Curriculum learning is a field in machine learning that employs a systematic approach that mimics human learning by organizing and sequencing the learning process. Unlike conventional machine learning methods that uniformly train models on randomly selected data points, curriculum learning introduces a structured curriculum or learning schedule that gradually exposes models to increasingly complex or informative examples.

In general, curriculum learning is a methodology to optimize the order in which experience is accumulated by the agent, to increase performance on a set of final tasks [1]. Recently, there has been much literature that combined curriculum learning with deep reinforcement learning agents to enhance the performances of the agents. For instance, metaheuristics algorithms are a popular class of methods to task sequencing problems (in other words, curriculum design) [1]. Many works such as [2] [3] introduced domain-specific novel heuristic algorithms. Automatic Curriculum Learning (ACL) has been successfully applied in various contexts, particularly in selecting environments from a discrete set. For example, ACL has been used to choose among different Minecraft mazes as demonstrated by Matiisen et al.[4], or to select levels in the game "Sonic the Hedgehog," as shown by Mysore et al.[5]. Further, a teacher-student type of method, which trains a separate learning agent to guide the student learning agent in parallel, was explored in [6] in a bipedal walker environment.

However, some of these aforementioned works require complex heuristics and significant human effort to design a successful curriculum, which often doesn't generalize well to a different environment or tasks. Others have a separate "teacher" agent (a Neural Network) that modifies the parameters of the environment during training time, to aid the learning process of the "student" agent. Pre-training the "teacher" agent or simultaneously training the "teacher" agent is nontrivial and adds another degree of freedom which may destabilize the learning process.

Recent advances in Large language model (LLM) and the accessibility of pre-trained LLM allowed researchers to apply LLM to machine learning problems. In the machine learning domain, heuristics design work by humans can be transferred to LLM, leveraging its ability to understand Natural Language, and context, and generalize. In curriculum learning, LLM has the potential to reduce human effort and level of intervention in the training process. In this work, we develop an algorithm for utilizing LLM for automatic curriculum design to enhance the performance and learning efficiency of a reinforcement learning agent.

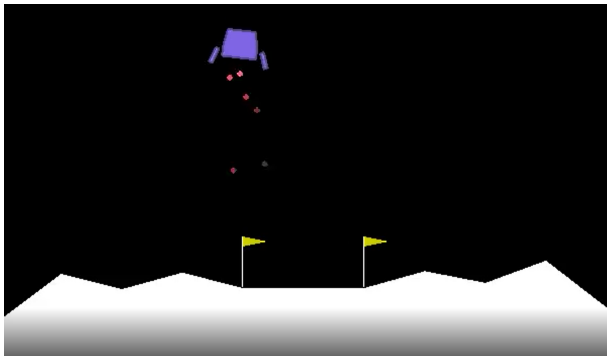

Fig. 1. Example of the LunarLander-V2 environment

II. Methodology

A. Environment

We tested our approach in LunarLander-V2 environment. The classic toy problem based on an OpenAI Box2D environment has a lander that aims to optimize its trajectory to land on the landing pad marked by the yellow flags as shown in Fig. 1. The action space consists of a discrete (binary) vector $a \in \mathcal{A}$ of size (2,), and the observation space consists of a hybrid state vector $s \in \mathcal{S}$ of size (8,). It contains the global coordinates of the lander's center of mass, linear velocities in the (x, y) directions, angle and angular velocity, and two booleans that represent whether each leg is in contact with the ground or not.

The episodic reward of the environment $R_{ep} \in \mathcal{R}_a$ is defined as follows

$$R_{ep} = \sum_{t}^{T_{ep}} (-0.3e_{main,t} - 0.03e_{side,t}) \\ 200b_{solved} + 100b_{rest} + 10b_{leg} - 100b_{crashed} \tag{1}$$

where $e_{main,t}$ and $e_{side,t}$ are booleans for firing the main and side engine for each frame, t. The t represents the number of

steps in an episode with the episode length of T_{ep}. b_{solved} are $b_{crashed}$ booleans for solving (landing on the landing pad) and crashing the lander. Finally, b_{leg} and b_{rest} correspond to the boolean for each leg with ground contact and lander coming to a rest. A well-trained agent is expected to have a reward greater than 200.

The environment has four parameters that change the transition dynamics $\mathcal{P}_a$ and hence the difficulty of the landing task. The parameters are $gravity, enable\_wind, wind\_power$, and $turbulence\_power$. In our experiments, we always set $enable\_wind$ to $True$. Hereon, we denote these parameters as $p_{Env} := [gravity, wind\_power, turbulence\_power]$. The bounds for the rest of the environment parameters are as follows:

$$\begin{aligned} -12.0 <& gravity < 0.0 \\ 0.0 \leq& wind\_power \leq 20.0 \\ 0.0 \leq& turbulence\_power \leq 2.0 \end{aligned} \tag{2}$$

By changing these environment parameters, the difficulty of the landing task can be adjusted. Given an environment and a landing task, we have an MDP=$(\mathcal{S}, \mathcal{A}, \mathcal{P}_a, \mathcal{R}_a)$ which is used in our reinforcement learning problem formulation.

B. Double Q-Learning

In our implementation, we used a Deep Q-Network (DQN) agent. The DQN agent approximates the Q-value function $Q(s, a)$, which measures the quality of an action in a given state, using a neural network [7]. By iteratively updating the network's weights to minimize a loss function that is proportional to the error between predicted and target Q-values, DQN aims to achieve more accurate estimations of optimal action-value pairs. To avoid overestimation bias in the critic update, we used a double-Q trick. For the details of the implementation for Double-Q learning please refer to Algorithm 1 and [8].

Algorithm 1 Double Q-Learning with LLM feedback

0: **procedure** TRAINDQN(env)
Require: env
  Initialize replay memory D
  Initialize the log
    Initialize primary Q-network parameters ϕ
    Initialize target Q-network parameters $\phi' = \phi$
    Set exploration rate ϵ and decay rate ϵ_{decay}
    Set C the target update period
    Set T the maximum training steps
    Set T_{eval} the evaluation period
    for episode $= 1$ to N_{ep} **do**
      $\mathrm{s}_1 = env.reset()$
    for $t = 1$ to T **do**
      With probability ϵ, select a random action a_t
      Otherwise, select $a_t = \arg\max_{a_t \in \mathcal{A}} Q_\phi(s_t, a_t)$
      Execute action a_t, observe r_t and s_{t+1}
      Store transition (s_t, a_t, r_t, s_{t+1}) in D
      Sample mini-batch of transitions (s, a, r, s') from D
      if terminal state s' **then**
        $y_j = r_j$
      else
        $y_j = r_j + \gamma Q_{\phi'}(s', {}_{a'}\, Q_\phi(s', a'))$
      end if
      $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i)^2$
      if $t \mod C == 0$ **then**
        $\phi' \leftarrow \phi$
      end if
      if $t \mod T_{eval} == 0$ **then**
        Evaluate the agent and save train_log
      end if
      $\epsilon \leftarrow \epsilon \times \epsilon_{\text{decay}}$
    end for
  end for
end procedure=0

C. Automatic Curriculum Learning with LLM

In the realm of reinforcement learning, the dynamic adjustment of training environments plays a pivotal role in enhancing the learning efficiency and robustness of agents. Our project innovates in this domain by integrating a Large Language Model (LLM) to serve as an automatic curriculum designer for a Lunar Lander agent. The LLM is tasked with generating and modifying environmental parameters—gravity, wind power, and turbulence power—based on the agent's performance metrics. This integration is a novel approach to curriculum learning, leveraging the LLM's capacity for complex decision-making and pattern recognition.

Prompt engineering is central to our methodology. It involves crafting structured inputs that guide the LLM in generating appropriate environmental parameters. The prompts are divided into several parts, each serving a distinct purpose, please refer to appendix V-B for more details of each prompt file:

- Initial System Prompt: This includes a basic description of the environment, the task's reward design, and the parameters (with their ranges) that need to be adjusted.
- Initial User Prompt: This is used to solicit an initial, straightforward output from the LLM.
- Code Output Tip: This assists the LLM in formatting its output correctly, ensuring it adheres to JSON format and uses appropriate numerical representations.
- Code Feedback: This is crucial for iterative learning. It guides the LLM to propose new parameter settings based on the agent's performance in previous training steps. It includes tips, constraints, and rules for analyzing metrics and adjusting the difficulty level.
- Policy Feedback: This incorporates metrics from the TensorBoard log, providing the LLM with data to inform its decisions for new parameter iterations.

The LLM is consulted periodically to determine whether to alter the environment's difficulty. If the LLM suggests

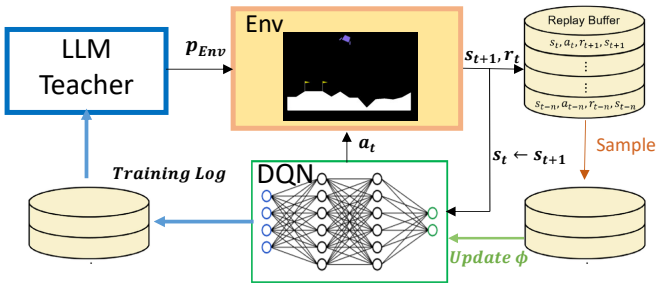

Fig. 2. Automatic Curriculum Learning with LLM Architecture

identical parameters to the current ones, training continues as is. However, if new parameters are proposed, the current training loop is terminated, and the environment is reset with these new settings. This process is repeated a predetermined number of times throughout the training phase.

Our decision-making criteria are grounded in a comprehensive analysis of various performance metrics, including critic loss, Q-values, and evaluation returns. These metrics, along with their mean, maximum, and minimum values, are logged in TensorBoard and subsequently downsampled to 10% of their original size to optimize token usage for the LLM. This downsampling is crucial for efficiently utilizing the LLM's capabilities within token constraints, ensuring that the most related information is considered in each decision-making cycle.

The algorithm for Automatic Curriculum Learning with LLM is presented in Algorithm 2 and visualized in Fig. 2. After initializing the environment using the LLM teacher, we iterate through N_{envs} environments with varying difficulties. LLM will read the training log of the DQN agent and decide to either keep the current environment and allow the agent to keep learning or change the environment with appropriately adjusted difficulty (p_{Env}).

Algorithm 2 Automatic Curriculum Learning with LLM

Require: LLM, $P_{feedback}$
Initialize LLM feedback period $P_{feedback}$
$p_{Env} \leftarrow$ LLM.init_generate()
Generate env $\leftarrow Env(p_{Env})$
for $n = 1$ to N_{envs} **do**
  while ¬change_env **do**
    train_log $\leftarrow$ TrainDQN(env)
    if train_log.steps mod $P_{feedback}$== 0 **then**
      $p'_{Env} \leftarrow$ LLM.iter_generate(train_log)
      env$' = Env(p'_{Env})$
    end if
    if $p_{Env} \sim= p'_{Env}$ **then**
      env $\leftarrow$ env$'$
      $p_{Env} \leftarrow p'_{Env}$
    end if
  end while
end for=0

III. Results

A. Training

In our experiment, we trained two models

1) DQN agent trained on set of environments $\mathcal{E}$ generated by ACL-LLM
2) DQN agent trained on the $\mathcal{E}$ from 1)

During the DQN agent's training procedure using the ACL-LLM Algorithm 2, the LLM Teacher generates N_{envs} distinct sets of environment parameters. Then, the DQN agent undergoes sequential training on these parameterized environments. This training paradigm dynamically adjusts the difficulty levels of the environments contingent upon the performance exhibited by the DQN agent. This adaptive adjustment process is designed to augment the generalization capabilities of the DQN agent. Then for Model 2, another DQN agent undergoes sequential training on the same N_{envs} environments, in a randomized order. The agent is trained in each environment for a fixed number of steps denoted as $P_{feedback}$. This is deliberately chosen to allow for a fair comparison as the Model 1 was trained for that same number of steps per environment. This is done to demonstrate the effectiveness of the LLM in generating a training curriculum by solving a generative and sequential problem of the training environments. We repeat the experiments for $P_{feedback}$ = 50K, 100K, and 300K to analyze the impact of longer training steps per environment. The experimental parameters are reported in Table I. The neural network architecture has 16 hidden units with 2 layers. For training batch size of 64 was used. We utilized the OpenAI gpt3.5-turbo model API as the backend of our LLM.

TABLE I
Experiment Parameters

Experiment	N_{envs}	N_{ep}	T	$P_{feedback}$	α	γ
1	10	1	1 M	50 K	0.001	0.99
2	10	1	1 M	100 K	0.001	0.99
3	10	1	1 M	300 K	0.001	0.99

We report the training results for $P_{feedback} = 100K$ in Fig. 3 and Fig. 4 for the DQN trained with ACL-LLM and baseline training algorithm correspondingly. The evaluation reward converges and fluctuates when the environment parameters are changed (indicated by the red dashed lines). Note that the evaluation rewards for the DQN trained with ACL-LLM is more stable than that of the baseline as the environment parameter changes are carefully selected by the LLM teacher whereas the environment parameter changes are random in the baseline training algorithm.

B. Evaluation Setup

In our study, the effectiveness of the ACL-LLM approach is evaluated against a baseline DQN agent as discussed above. For evaluation purposes, both agents are subjected to a set of environments generated with uniformly sampled parameters. This approach is intended to test the agents' performance in unseen scenarios, thereby assessing their generalization capabilities. The assumption underpinning this methodology

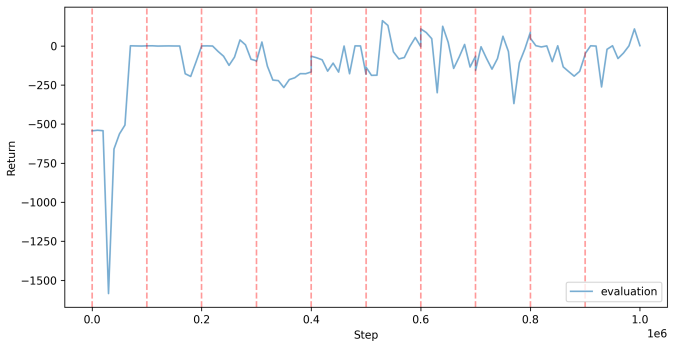

Fig. 3. Average Evaluation Reward of a DQN agent trained with ACL-LLM in the LunarLander-v2 environment during the training phase for $P_{feedback} = 100K$. The red dashed lines indicate where the environment has changed.

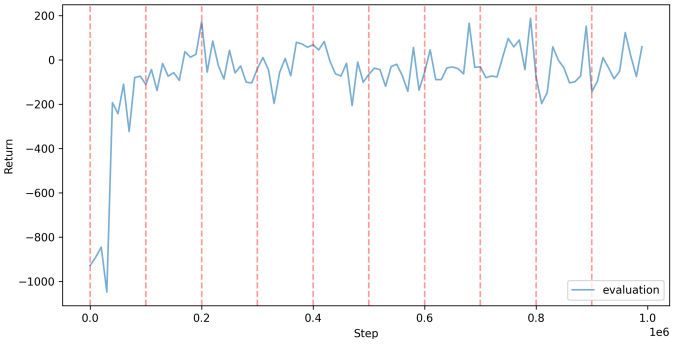

Fig. 4. Average Evaluation Reward of a DQN baseline agent in the LunarLander-v2 environment during the training phase for $P_{feedback} = 100K$. The red dashed lines indicate where the environment has changed.

is that a well-designed curriculum, as proposed by the LLM, not only enhances learning efficiency but also equips the agent with a more robust and versatile skill set, enabling it to perform effectively in a wider range of scenarios.

We run evaluations in 10 different environments with 100 trajectories per environment.

C. Evaluation Results

For $P_{feedback} = 50K$, the effects of training with a LLM-generated curriculum are apparent as shown in Fig. 5. While both models do not land successfully on average, the model trained with ACL-LLM performs strictly better (by 42.97 on average) than the DQN baseline model. Also, in some test environments, a model trained with ACL-LLM does succeed whereas DQN baseline model fails in every test environment.

For $P_{feedback} = 100K$, similar behaviors are observed as in $P_{feedback} = 50$ case. While both models do not land successfully on average, the model trained with ACL-LLM performs strictly better (by 78.44 on average) than the DQN baseline model. As shown in the maximum return in Fig. 6, both models can land on the landing pad at least once out of 100 evaluation trajectories across all test environments. However, the model trained with ACL-LLM Algorithm performs strictly better than the DQN baseline model across all test environments.

Lastly, for $P_{feedback} = 300K$, the model trained with ACL-LLM no longer performs better than the DQN baseline model (-32.14 on average). We suspect that the effectiveness of training with curriculum has diminished at this point as the DQN baseline model can train for sufficient steps in each training environment. Also, we note that the task in LunarLander-V2 environment is not too complex. Therefore, with sufficient training duration, a DQN agent can perform well in evaluation. We hypothesize that the ACL-LLM algorithm will be more effective in more complex task, such as Parkour-v0 environment in Appendix V-A. The results demonstrate that ACL-LLM can enhance evaluation performance even with small training steps for tasks where training is expensive.

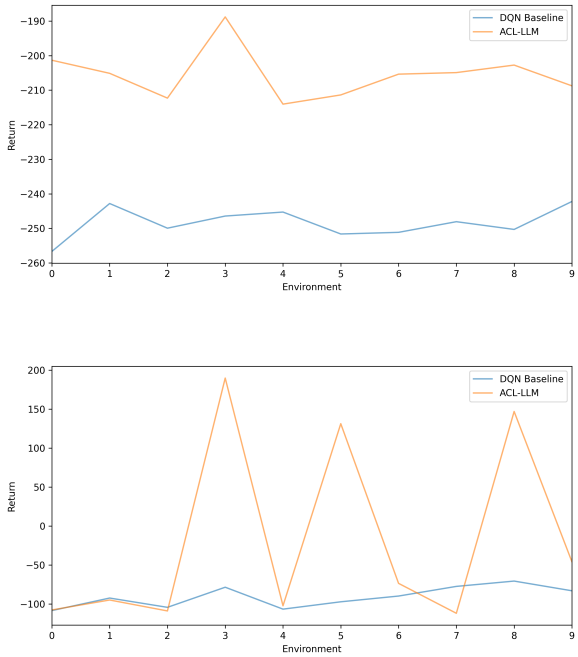

Fig. 5. Average evaluation return (Top) and max return (Bottom) on 10 randomly generated environments using the model trained with $P_{feedback}$=50K

IV. Conclusion

In this work, we presented a novel automatic curriculum learning algorithm leveraging the large language model's capacity for contextual understanding and pattern recognition to replace human labor in curriculum design. In smaller training steps, the ACL-LLM demonstrated superior performances in the test evaluation compared to the DQN agent trained for the same number of training steps per training environment. This showed the effectiveness of the curriculum (sequence of environments with varying difficulty) generated by the LLM. However, for higher training steps, the effectiveness of the curriculum generated by the LLM diminished. Therefore, for tasks where training is expensive, ACL-LLM algorithm can enhance the learning process by automatically generating a curriculum that can enhance the generalizability of the trained agent.

For future work, we would like to train and evaluate the ACL-LLM algorithm in a more complex and difficult task such

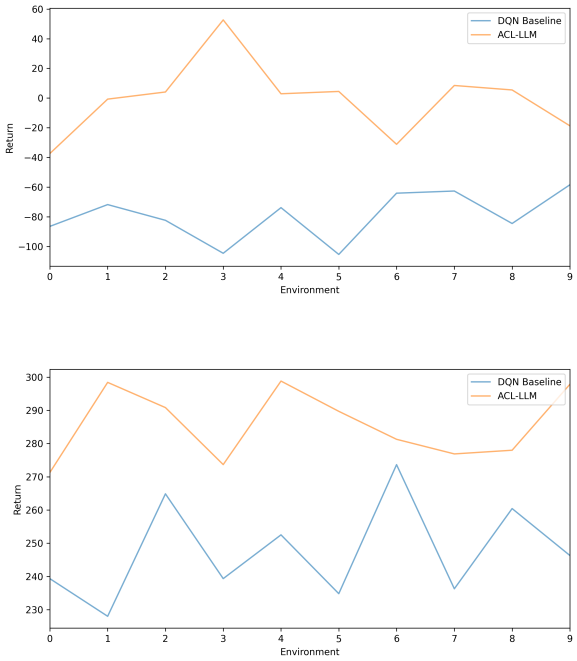

Fig. 6. Average evaluation return (Top) and max return (Bottom) on 10 randomly generated environments using the model trained with $P_{feedback}$=100K

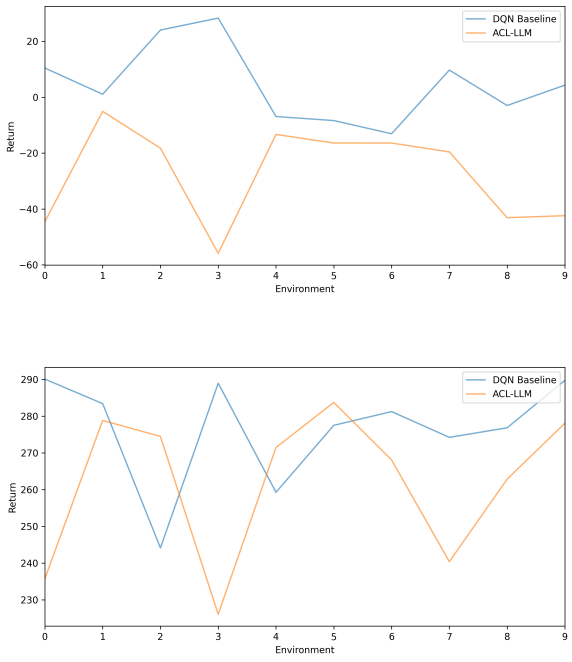

Fig. 7. Average evaluation return (Top) and max return (Bottom) on 10 randomly generated environments using the model trained with $P_{feedback}$=300K

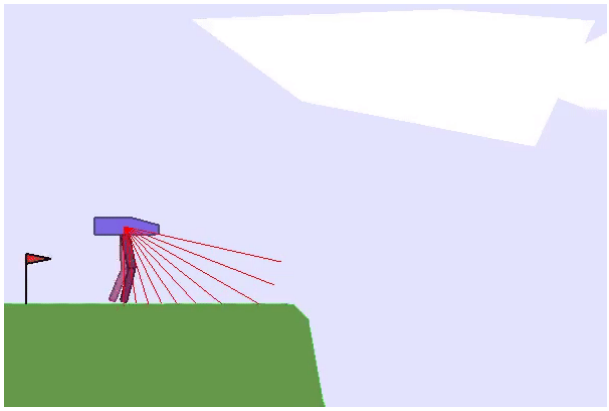

Fig. 8. Example of the Parkour-v0 environment

as Parkour-v0 environment described in Appendix V-A. From this, we would be able to draw a more holistic conclusion on the effectiveness of our proposed algorithm.

V. Appendix

We would like to note that our team initially started with three team members. However, one team member decided to drop the course a few weeks after the milestone report submission. Since then, we consulted with the GSIs and decided to change the topic and scope of the project to be more fitting for a group of two people.

Also, we wanted to note what we have attempted but aborted due to a lack of time and resources in the appendix.

A. Environment

Initially, we wanted to test our approach on the custom Parkour-v0 environment [6] as shown in Fig. 8, which is based on the BipedalWalker-v2 OpenAI gym environment. This custom gym environment consists of various terrains and a bipedal walker agent that aims to reach the end of the terrain. The difficulty of the terrain depends on the roughness of the ground surface, the number of hills, and the steepness of the hills. The observation vector is a continuous vector of shape (36,), and the action space is a continuous vector of size (4,). Observation consists of lidar sensor measurements (distance), head positions, and joint positions [6]. The action represents the torque applied on 4 joints of the bipedal walker agent (e.g. 2 hip joints and 2 knee joints).

The base environment utilizes a pre-trained Compositional pattern-producing network (CPPN) to generate terrains with varying roughness and difficulty [6]. We modified the base environment to replace the CPPN with a large language model (LLM) to generate the terrain during training to enable LLM to directly generate an environment. While the Algorithm 2 runs in this environment, the training of a Soft-Actor-Critic (SAC) agent in this environment was unsuccessful and long. For instance, Fig. 9 shows the average evaluation reward on a single relatively flat and easy terrain, which took about **7 hours** to run. The converged average reward is about -15 whereas a successful walker should have a reward greater than 250. If we

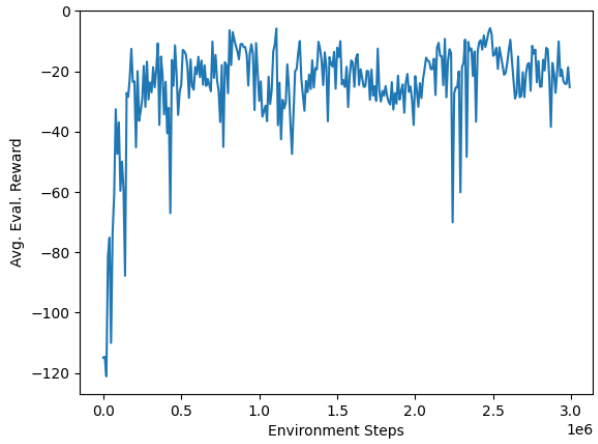

Fig. 9. Average Evaluation Reward of a SAC agent in the Parkour-v0 environment. A successful agent should achieve a reward of about 230

wanted to run ACL-LLM for this setup, it would take about **72 hours** for $N_{envs} = 10$, including the LLM query time. After careful consideration, we ultimately aborted this environment as time and manpower were limited to tune the SAC agent.

B. LLM Prompts

1) Initial User: You are an environment designer for Lunar Lander.

Some key information about the Lunar Lander environment are:

- The lander starts at the top center of the viewport with a random initial force applied to its center of mass.
- Reward for moving from the top of the screen to the landing pad and coming to rest is about 100-140 points. If the lander moves away from the landing pad, it loses reward. If the lander crashes, it receives an additional -100 points. If it comes to rest, it receives an additional +100 points. Each leg with ground contact is +10 points. Firing the main engine is -0.3 points each frame. Firing the side engine is -0.03 points each frame. Solved is 200 points.

The episode finishes if:

- The lander crashes (the lander's body gets in contact with the moon)
- The lander gets outside of the viewport (x coordinate is greater than 1)
- The lander is not awake. From the Box2D docs, a body which is not awake is a body which doesn't move and doesn't collide with any other body

The user will use your environment to train a RL agent (an Deep Q-Network Agent). You should return 3 different values

- gravity dictates the gravitational constant. It is strictly less than 0.0 and strictly greater than -12.0.
- wind_power dictates the maximum magnitude of linear wind applied to the craft. It is strictly less than 20.0 and strictly greater than 0.0.
- turbulence_power dictates the maximum magnitude of rotational wind applied to the craft. It is strictly less than 2.0 and strictly greater than 0.0.

2) Initialize System: Please give me a set of environment parameters to start training my Lunar Lander.

3) Code Output Tips: Below are some constraints that you should follow when return the values.

- Return only 1 set of parameters
- Return your value in a json format only, like the following example:
  {"gravity":-0.1,"wind_power":0.1,"turbulence_power":0.1}

4) Code Feedback: Please carefully analyze the policy feedback and provide a new environment setting. Please follow the following rules. You should not change the parameters that you gave in the past round if the agent is still learning, unless:

- The performance of the agent is good and the eval_return has converged. Make the environment harder.
- The performance of the agent is not increasing at all, showing that it is not learning. Make the environment easier.

Some tips for analyzing the performance:

- Maximum possible eval_return is 200
- If the eval_return converges fast, you should make the environment harder.
- If the eval_return is fluctuating without increasing, you should make the environment easier.

Some helpful tips for altering difficulty of the environment:

- You can make the environment harder by increasing the gravity.
- You can make the environment harder by increasing the wind_power.
- You can make the environment harder by increasing the turbulence_power.

5) Policy Feedback: We trained a RL policy using the provided terrains and tracked global policy metrics such as evaluation return and episode lengths after every epoch_freq epochs and the maximum, mean, minimum values encountered:

REFERENCES

[1] Sanmit Narvekar et al. "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey". In: *J. Mach. Learn. Res.* 21.1 (Jan. 2020). ISSN: 1532-4435.

[2] Carlos Florensa et al. *Reverse Curriculum Generation for Reinforcement Learning*. 2018. arXiv: 1707.05300 [cs.AI].

[3] Lu Jiang et al. "Self-Paced Curriculum Learning". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 2694–2700. ISBN: 0262511290.

[4] Tambet Matiisen et al. "Teacher-Student Curriculum Learning". In: *CoRR* abs/1707.00183 (2017). arXiv: 1707.00183. URL: http://arxiv.org/abs/1707.00183.

[5] Siddharth Mysore. “Reward-guided Curriculum for Robust Reinforcement Learning”. In: 2019. URL: https://api.semanticscholar.org/CorpusID:197631497.

[6] Clément Romac et al. “TeachMyAgent: a Benchmark for Automatic Curriculum Learning in Deep RL”. In: *CoRR* abs/2103.09815 (2021).

[7] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (Feb. 2015), pp. 529–33. DOI: 10.1038/nature14236.

[8] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. 2015. arXiv: 1509.06461 [cs.LG].