Improving the Data-efficiency of Reinforcement Learning by Warm-starting with LLM

Thang Duong
University of Arizona
thangduong@arizona.edu

Minglai Yang
University of Arizona
mingly@arizona.edu

Chicheng Zhang
University of Arizona
chichengz@cs.arizona.edu

Abstract

We investigate the usage of Large Language Model (LLM) in collecting high-quality data to warm-start Reinforcement Learning (RL) algorithms for learning in some classical Markov Decision Process (MDP) environments. In this work, we focus on using LLM to generate an off-policy dataset that sufficiently covers state-actions visited by optimal policies, then later using an RL algorithm to explore the environment and improve the policy suggested by the LLM. Our algorithm, LORO, can both converge to an optimal policy and have a high sample efficiency thanks to the LLM's good starting policy. On multiple OpenAI Gym environments, such as CartPole and Pendulum, we empirically demonstrate that LORO outperforms baseline algorithms such as pure LLM-based policies, pure RL, and a naive combination of the two, achieving up to $4\times$ the cumulative rewards of the pure RL baseline.

1 Introduction

The standard protocol in online RL has many interesting applications, from playing games Silver et al. [2017] to robotic control Kober et al. [2013]. While having impressive empirical performance and enjoying the theoretical guarantee on returning the optimal policy under some assumptions Ramaswamy and Hüllermeier [2021], Agarwal et al. [2019], Bertsekas [2007], a key problem of this approach is the notorious sample inefficiency, which limits its application in practice. Thus, most impressive successes in online RL have been restricted to settings where many samples can be obtained by interacting with the environment (such as games or environments with high-quality simulations).

To overcome this, Lange et al. [2012], Ernst et al. [2005], Riedmiller [2005], and Levine et al. [2020] proposed the Offline RL setting, where the algorithm does not directly interact with the environment as in online RL, but is trained on a large dataset of experience collected from some other sources (e.g., by expert demonstration). While the sample complexity problem is mitigated due to the large training dataset, these Offline RL methods suffer from the *distribution shift* problem, where the state distribution from the offline data differs significantly from the one induced by online interactions Wang et al. [2021].

A popular approach to address the distribution shift problem is by aggregating both the offline and online data Xie et al. [2021], Song et al. [2022], Zhang and Zanette [2023] This offline-to-online approach greatly reduces the sample complexity of the problem by reducing unnecessary exploration with the offline dataset while also mitigating the distribution shift problem through online interactions. Under some assumptions, Song et al. [2022] provides a cumulative regret and sample complexity guarantee for the offline-to-online setting. They show that, if the offline data distribution covers some high-quality policies' trajectories, their offline-to-online algorithm is both sample-efficient and competitive with the high-quality policies covered by the offline data. Even then, our goal is to further

First Exploration in AI Today Workshop at ICML (EXAIT at ICML 2025).

improve the sample efficiency, perhaps by leveraging extra information from the problem description and common sense (e.g., avoid obstacles, find the key to open the door, etc.)

Recently, LLM has shown a remarkable ability for reasoning, even in hard sequential decision problems such as robot manipulation Ahn et al. [2022], Huang et al. [2022], Liang et al. [2023]. Still, Carta et al. [2023] claims that, while useful, LLM's application for sequential decision tasks is limited by the absence of environment grounding. They demonstrate this in an interactive textual environment and show that online interaction with the environment can help "ground" the LLM and boost the overall performance. Thus, we raise the question:

Can we design an algorithm that can both converge to the optimal policy and has high sample efficiency by leveraging the reasoning capability of LLMs?

In this paper, we answer this question positively. Under Assumption 1, where the policy suggested by the LLM has sufficient coverage of an optimal policy, our algorithm, LLM Off-policy pretrain, RL On-policy (LORO), enjoys both small Cumulative suboptimality and Sample complexity as suggested by Song et al. [2022]. To the best of our knowledge, we are the first to suggest warm-starting RL with LLM's collected data and connect LLM with offline-to-online RL, drawing the similarity between the distribution shift problem in Offline RL versus the useful-but-suboptimal policy extracted from LLM, and suggest that the offline-to-online RL approaches can be applied here. Empirically, we demonstrate the effectiveness of our algorithm in multiple OpenAI Gym environments Towers et al. [2024] compared to other baselines such as pure LLM-based policies, pure RL, and a naive combination of the two.

2 Related work

Offline-to-online RL: Nair et al. [2020] showed that a naive combination of offline pre-training and online fine-tuning does not usually help and often worsen the performance, a large part due to excess conservatism. Many more sophisticated approaches have been studied empirically: Nair et al. [2018], Hester et al. [2018], Uehara et al. [2021], Ball et al. [2023]. However, none of these works studied the utility of LLMs in warm-starting online RL. In contrast, our goal is to improve the sample efficiency further by combining this approach with LLM, especially if the LLM can understand the problem description and use common sense to take good actions.

Warm-starting RL: Schmitt et al. [2018] propose to kick-start Deep RL with a teacher policy by adding an extra objective to encourage the learner to behave similarly to the teacher, with a diminishing weight to allow the student to eventually surpass the teacher. One limitation is that Schmitt et al. [2018] assumes the teacher policy is high-performing enough enough to be distilled, meaning its application is limited when learning a new task from scratch. In contrast, we only require the initial policy to sufficiently cover the state-action pairs often visited by an optimal policy. This is a much milder assumption and is reflected in our Experiment section, where a very weak initial LLM policy can still be useful.

Embodied LLM and environment interactions: Recently, LLM has showed very impressive capability Brown et al. [2020], including understanding about physics Patel and Pavlick [2022] Liu et al. [2024], color Abdou et al. [2021], and affordances between bodies and object Ahn et al. [2022]. This implicit knowledge could be the reason why LLM can be used to directly manipulate robots Ahn et al. [2022], Huang et al. [2022], Liang et al. [2023]. However, Carta et al. [2023] claims that LLMs lack grounding due to 1) the training objective of next word prediction not aligned with other goals, and 2) no interactions with the environment.

Many works seemingly agree with Carta et al. [2023] and incorporate environment interactions, thus showing significant improvement. A popular approach is letting the LLM interact directly with the environment and collect the feedback for the subsequent prompt Carta et al. [2023], Yao et al. [2022], Zhou et al. [2023], Luketina et al. [2019]. Another direction is a two level system, where the LLM take high level, abstract actions (such as creating sub-goals Bhat et al. [2024] Dalal et al. [2024] or choosing the skills to use Liang et al. [2023], Ahn et al. [2022]), and the low level classical system implementing the LLM's "plan" in practice. A related work from Hao et al. [2024] uses LLM to extract and formulate the problem's objectives, constraints, and may include sub-goals creation, for the low-level optimization solver.

Theoretical analysis on LLM's exploration in MDPs: Many works investigate how LLM performs compared to traditional methods, such as UCB, in MDP problems. Arumugam and Griffiths [2025] introduces a more explicit method for exploration using Posterior Sampling. Chen et al. [2024] uses LLM to construct multiple policies and combine with a model selection algorithm to solve Contextual Bandit. Nie et al. [2024], Krishnamurthy et al. [2024] investigate how LLM explores in the Bandit problem and show that the base LLM policies are non-trivial, but sub-optimal. This assessment aligns with our experiment results.

Using LLM to provide extra information for RL: Carta et al. [2023] and Tan et al. [2024] use LLM directly to generate the policy and fine-tune it with RL (using Policy Gradient with PPO or an Actor-Critic framework). Lee et al. [2023] and Lin et al. [2023] propose pretraining an LLM with an offline dataset and show that it can both explore online and act conservatively offline. Unlike them, instead of an end-to-end approach that mixes the RL objective (of maximizing the cumulative reward) with the LLM objective (for next token prediction), we have a separate, smaller RL learner trained exclusively on the classical RL objective that enjoys the typical asymptotic optimality. Since one of our motivations is computational efficiency, training a large neural network that requires a lot of data would defeat the point of using LLM to help reduce the sample complexity. Zheng et al. [2022] pre-trains a transformer-based neural network on the offline dataset and develops a way to efficiently fine-tune it with online interaction. This differs from our proposal since we don't have an offline dataset, but the data collected by the LLM's policy can be regarded as a small offline dataset. Another closely related work is Du et al. [2023], where the LLM guides the algorithm's exploration by generating (sub) goals and rewards the RL algorithm when achieving these goals. While both this work and ours leverage LLM to reduce unnecessary exploration for RL, they focus more on sub-goal generation and providing intrinsic reward in sparse feedback problems, while we are focusing on dense reward settings where RL online interactions can refine the warm-started but sub-optimal policy given by the LLM. Finally, Choi et al. [2022] and Kant et al. [2022] use LLM to provide a prior for the policy to help the learner explore more efficiently, which is similar to our motivation on a high level.

Other ways LLM can help solving MDPs: Besides low-level control and high-level planning, Jeong et al. [2024] also investigates how LLM can help robot intelligence systems by reward design (to combine with RL) Ma et al. [2023], Xie et al. [2023], and scene understanding Huang et al. [2023], Hong et al. [2023]. Even with these successes, there are still many challenges in deploying LLM to solve sequential decision problems in practice, such as the lack of a guarantee of finding the optimal solution.

3 Preliminaries

Consider a Markov Decision Process $M(\mathcal{S}, \mathcal{A}, T, H, R, P, d_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, T is the number of episode, H is the horizon of each episode, the reward function is $R(s,a) \in \Delta([0,1])$ and the transition dynamic $P(s,a) \in \Delta(\mathcal{S})$ at (s,a), and $d_0(\mathcal{S})$ is the initial distribution. At each step h, the learner chooses from its policy π an action $a_h \leftarrow \pi(s_h)$ and receive the reward from the Reward function: $r_h \leftarrow R(s_h, a_h)$, and transitions to the next state $s_{h+1} \sim P(s_h, a_h)$. The optimal policy π^* is defined as a policy that has a maximum expected cumulative reward: $\pi^* = \operatorname{argmax}_\pi \mathbb{E}\left[\sum_{h=1}^{H} r_h \mid \pi\right]$. We also have access to an initial policy π_{LLM} that satisfies Assumption 1. The goal is to maximize the cumulative reward by making use of π_{LLM} to improve the sample efficiency.

Assumption 1. *Following π_{LLM} can produce trajectories with state-action pairs that sufficiently cover an optimal policy of the MDP.*

Assumption 1 states that an LLM can zero-shot suggest non-trivial base policies, but they are not optimal. We see an analogous phenomenon with distribution shift in offline RL that results in a suboptimal policy. Thus, we hypothesize that aggregated trajectories collected with LLM, which avoid trivial state-action data (such as unnecessary repetitions, visiting absorbing states, etc), and refine the learned policy later with online interaction, as the offline-to-online protocol, can be useful. Assumption 1 enables the LLM's collected dataset to satisfy Song et al. [2022], thus allowing our algorithm to enjoy their Cumulative suboptimality regret and Sample complexity guarantees.

Algorithm 1 LLM Off-policy pre-train, RL On-policy (LORO)

1: **Input:** # of episodes T, # of LLM data collection episode τ, episode length H, off-policy RL algorithm $\mathrm{Alg}(\cdot)$
2: **Initialize:** LLM off-policy dataset: $\mathcal{D} = \{\emptyset\}$
3: **for** $t = 1, \cdots, \tau$ **do** ▷ LLM data collection
4:     **for** $h = 1, \cdots, H$ **do**
5:         Observe state s_h^t, take action $a_h^t \leftarrow \pi_{\mathrm{LLM}}(s_h^t)$, and receive reward r_h^t
6:         Aggregate data $\mathcal{D} \leftarrow (s_h^t, a_h^t, r_h^t)$
7:     **end for**
8: **end for**
9: Pre-train the policy $\pi_{prev} \leftarrow \mathrm{Alg}(\mathcal{D})$
10: **for** $t = \tau + 1, \cdots, T$ **do** ▷ Online learning
11:     **for** $h = 1, \cdots, H$ **do**
12:         Get policy $\pi_h^t \leftarrow \mathrm{Alg}(\mathcal{D}, \pi_{prev})$ ▷ Online updating the policy with the new data
13:         Observe state s_h^t, take action $a_h^t \leftarrow \pi_h^t(s_h^t)$, receive reward r_h^t
14:         Aggregate data $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_h^t, a_h^t, r_h^t)\}$
15:         Update $\pi_{prev} = \pi_h^t$
16:     **end for**
17: **end for**

4 The LLM Off-policy pre-train, RL On-policy (LORO) Algorithm

Our LORO algorithm is very straightforward. Under Assumption 1, the policy π_{LLM} collects high-quality data from the region that an optimal policy often visits. By only focusing on this and not exploring the low-quality data regions that are avoided by all optimal policies (*e.g.*, hitting the wall, absorbing states, etc.), we can significantly increase the sample efficiency. Thus, we use π_{LLM} to collect a small amount of data, off-policy "pre-train" a policy π with it, and then do on-policy online learning to fine-tune π to be optimal with a much smaller number of observations.

The details of our algorithm, LORO, are shown in Figure 1 and Algorithm 1. Initially, we use the LLM policy π_{LLM} to collect data for the first τ episodes (line 3 - 8). Then, we pre-train a policy using a classical RL algorithm on the data collected by LLM (line 9). Finally, we online fine-tune the pre-trained policy (line 10 - 17).

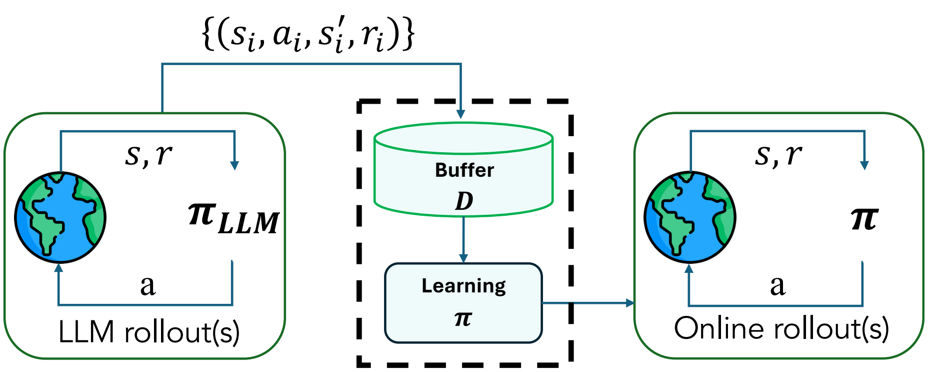

Figure 1: The LLM Off-policy pre-train, RL On-policy (LORO) algorithm. Image inspired by Levine et al. [2020].

5 Experiment

We empirically evaluate our algorithm[1] on a host of RL environments: Cart Pole, Pendulum, Frozen Lake, Cliff Walking, Represented Pong, and Mountain Car. We defer the environments' descriptions and RL implementation details to Appendix A.1, and the LLM setup to Appendix B.

[1]The code of our experiment can be viewed at `https://anonymous.4open.science/r/LlamaGym-551D/README.md`

Here, we compare our algorithm with these baselines:

- **On-policy**: a classical on-policy RL method, in which the algorithm collects the data and refines its policy in an online manner.
- **LLMs as Policies (Qwen-7B-Instruct, Qwen-32B-Instruct)**: the base policies from the 7B and 32B of the Qwen 2.5 series with Instruction tuning Yang et al. [2024]. For each episode t and step h, the LLM has access to the environment and observation descriptions s_h^t, and the action a_h^t is taken using Chain-of-Thought Wei et al. [2022]. The LLM setup details are in Appendix B. The prompt setup and examples are in Appendix C. Note that we only show the average episode reward collected in the first τ episodes, $r_{\text{avg}} = \frac{1}{H\tau}\sum_{t=1}^{\tau}\sum_{h=1}^{H} r_h^t$, in the figures below.
- **Random**: a policy π_{random} that take uniformly random action a_h^t. Similarly, we only show the average episode reward collected in the first τ episodes.

In the experiments below, we choose $\tau = 10$ and the number of pre-training steps is 1000. The task length T is 150 for CartPole, FrozenLake, 200 for CliffWalking, Pendulum, RepresentedPong, and 300 for MountainCar. LORO is trained using the data collected by Qwen-7B. In addition, we explore the effects of LLM model size, and the number of pre-training steps and varying the amount of LLM data τ, with results provided in Appendix A.3, A.4, and A.5.

5.1 Main results

The main results of our algorithm are shown in Figure 2.

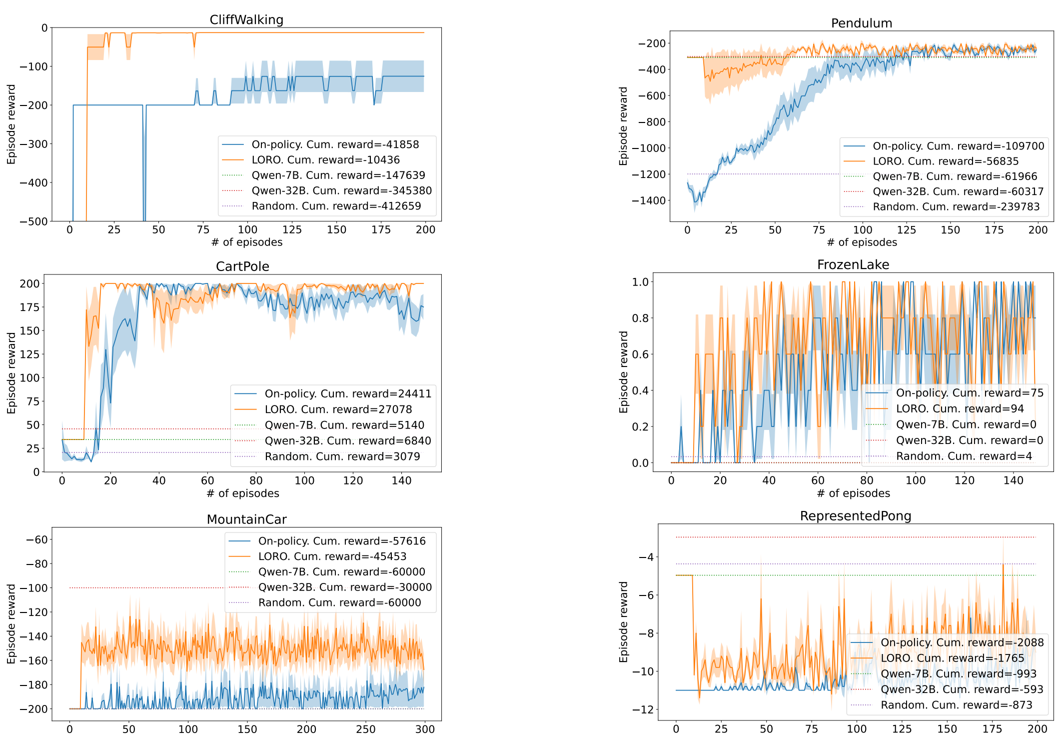

Figure 2: Our algorithm, LORO, outperforms the LLM policies (Qwen 7B, Qwen 32B) and the vanilla On-Policy RL baselines. In these six environments, the data required to learn the optimal policy for LORO is reduced from two to ten times the vanilla On-Policy baseline. LORO and the On-policy baseline learn the optimal policy in the first four environments. Even when not converged to the optimal solution, LORO outperforms other baselines in the last two environments. All results are shown with standard error over five random seeds. In the CliffWalking experiment, some baselines are not shown in the figure since their episode rewards are too small. Similarly, multiple baselines overlap at -200 on the MountainCar experiment.

5.2 Effect of pre-training

In this section, we would like to verify the importance of pre-training (Algorithm 1 line 9) for LORO's performance. In Figure 3, we show that mixing the data alone (which is equivalent to Song et al. [2022]) is insufficient. Our conjecture is that the pre-training step "overfits" LORO using only the high-quality data without the data from regions less visited by the optimal policy. As shown in Figure 2, pre-training significantly boosts the performance of LORO compared to just mixing the LLM's collected data with the on-policy collected data after τ episodes.

Even though pre-training can be useful initially, to behave optimally, the agent still needs to explore other state-action pairs in case the initial data comes from a sub-optimal policy, as shown in the CartPole environment in Figure 2.

At first sight, there seems to be a contradiction between our findings and Song et al. [2022]. Song et al. [2022] assumes access to a large offline dataset and, along with Nair et al. [2020], wants to keep the policy less conservative toward the offline data by treating the online versus offline data equivalently. In contrast, we don't have access to offline data. We instead use LLM to collect a small number of high-quality data, thus, unlike Song et al. [2022], LORO puts more weight on these observations initially. Our experiment showed that being conservative by "overfitting" to the LLM dataset can help learning more efficiently.

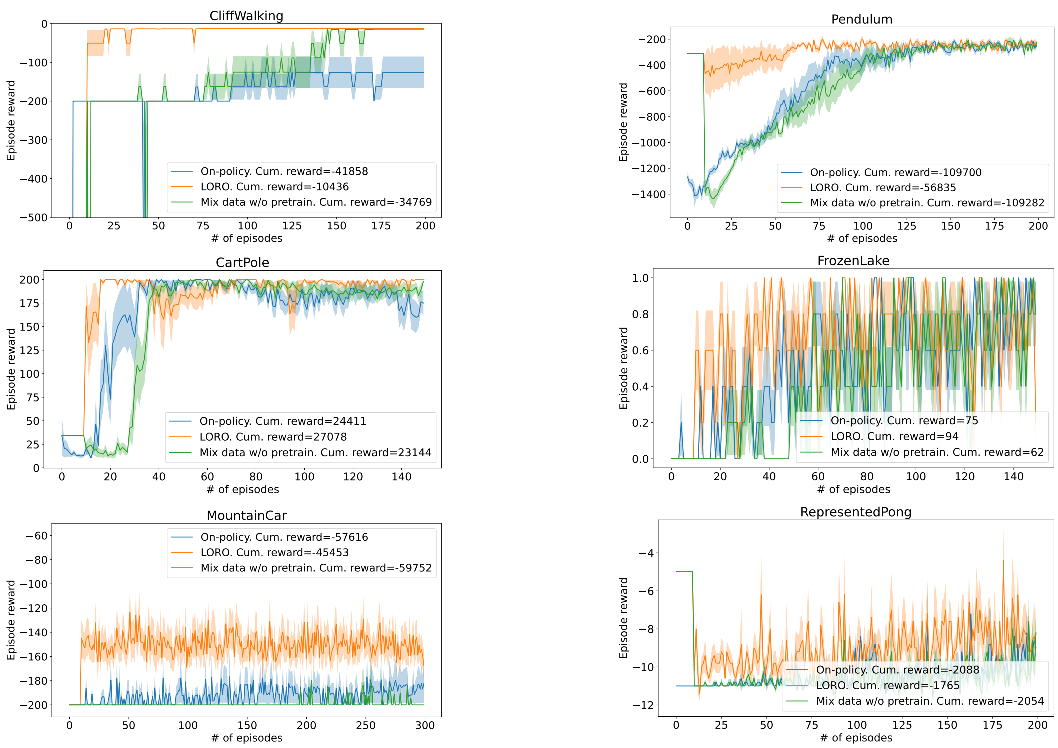

Figure 3: Comparing pre-training (then removing the collected data) versus mixing the LLM's collected data with on-policy data without pre-training. It's clear that pre-training is necessary for LORO to achieve superior performance compared to naively mixing the data.

5.3 Effect of LLM's data

In the previous section, Figure 3 shows us the importance of pre-training using data collected by LLM. In this section, we perform an ablation study that demonstrates that the quality of such data is crucial. In Figure 4, we show that using LLM's collected data is significantly better than using data collected with an On-policy RL algorithm from scratch or a policy that takes actions uniformly at random. Thus, we conclude that pre-training is only beneficial when coupled with high-quality data, which supports our conjecture above.

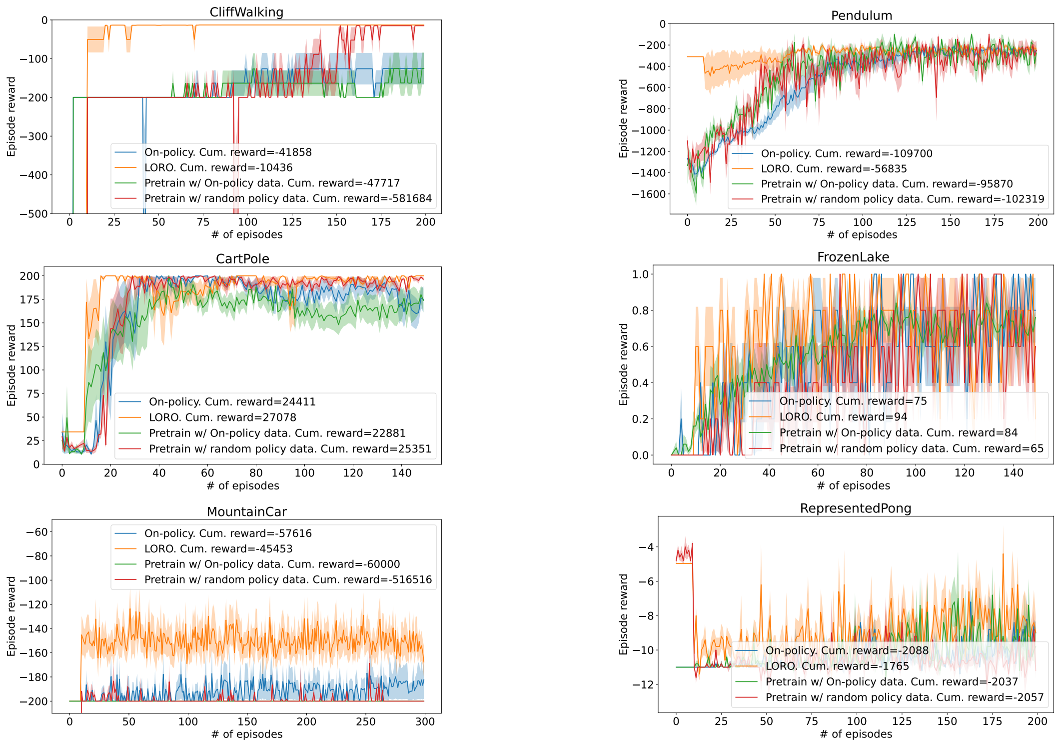

Figure 4: Comparing pre-training with LLM's data versus random and on-policy data. The main finding here is that pre-training is only useful with LLM's data.

5.4 Other findings

The effect of LLM's reasoning capability on the task's performance is shown in Appendix A.2. Overall, there is no clear correlation between the improvement in LLM's reasoning capability and LORO's decision-making performance. In addition, we find no clear relationship between the task's performance and the number of pre-training steps or the model size. These are shown in Appendix A.4, A.3. We also found no clear difference between environments with Discrete Action versus Continuous Action (e.g., Pendulum), despite the intuition that the Discrete Action environments should be easier for the LLM Singh et al. [2025].

6 Conclusion and Future work

In this paper, we investigate how to leverage an LLM to warm-start traditional RL methods. Empirically, we have shown that the high-quality data collected by the LLM can significantly increase the sample efficiency of online RL. Our algorithm, under Assumption 1, is supported by the Cumulative suboptimality regret and Sample complexity guarantee from Song et al. [2022].

Our work provides a framework for significantly reducing the sample complexity in RL problems, aiming toward practical applications where the data collection cost or safety is a major concern. A limitation of our work is that Assumption 1 may not hold for some RL tasks, but we believe that the increasing capability of LLM would increase the range of problems where Assumption 1 is applicable. In the future, we would like to extend this work to more sophisticated RL problems, with a large State and Action space. We would also like to investigate how to scale the sample efficiency with the LLM's capability.

References

Mostafa Abdou, Artur Kulmizev, Daniel Hershcovich, Stella Frank, Ellie Pavlick, and Anders Søgaard. Can language models encode perceptual structure without grounding? a case study in color. arXiv preprint arXiv:2109.06129, 2021.

Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. Reinforcement learning: Theory and algorithms. CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep, 32:96, 2019.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. arXiv preprint arXiv:2204.01691, 2022.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari. arXiv preprint arXiv:1906.08226, 2019.

Dilip Arumugam and Thomas L. Griffiths. Toward efficient exploration by large language model agents, 2025. URL `https://arxiv.org/abs/2504.20997`.

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data, 2023. URL `https://arxiv.org/abs/2302.02948`.

Dimitri P Bertsekas. Neuro-dynamic programming: An overview and recent results. In Operations Research Proceedings 2006: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Jointly Organized with the Austrian Society of Operations Research (ÖGOR) and the Swiss Society of Operations Research (SVOR) Karlsruhe, September 6–8, 2006, pages 71–72. Springer, 2007.

Vineet Bhat, Ali Umut Kaypak, Prashanth Krishnamurthy, Ramesh Karri, and Farshad Khorrami. Grounding llms for robot task planning using closed-loop state feedback. arXiv preprint arXiv:2402.08546, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In International Conference on Machine Learning, pages 3676–3713. PMLR, 2023.

Dingyang Chen, Qi Zhang, and Yinglun Zhu. Efficient sequential decision making with large language models. arXiv preprint arXiv:2406.12125, 2024.

Kristy Choi, Chris Cundy, Sanjari Srivastava, and Stefano Ermon. Lmpriors: Pre-trained language models as task-specific priors. arXiv preprint arXiv:2210.12530, 2022.

Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, and Ruslan Salakhutdinov. Plan-seq-learn: Language model guided rl for solving long horizon robotics tasks. arXiv preprint arXiv:2405.01534, 2024.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In International Conference on Machine Learning, pages 8657–8677. PMLR, 2023.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. Journal of Machine Learning Research, 6, 2005.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019. URL `https://arxiv.org/abs/1812.05905`.

Yilun Hao, Yang Zhang, and Chuchu Fan. Planning anything with rigor: General-purpose zero-shot planning with llm-based formalized programming. arXiv preprint arXiv:2410.12112, 2024.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.

Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3d-llm: Injecting the 3d world into large language models. Advances in Neural Information Processing Systems, 36:20482–20494, 2023.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL `https://arxiv.org/abs/2106.09685`.

Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation. In 2023 IEEE International Conference on Robotics and Automation (ICRA), pages 10608–10615. IEEE, 2023.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. arXiv preprint arXiv:2207.05608, 2022.

Hyeongyo Jeong, Haechan Lee, Changwon Kim, and Sungtae Shin. A survey of robot intelligence with large language models. Applied Sciences, 14(19), 2024. ISSN 2076-3417. doi: 10.3390/app14198868. URL `https://www.mdpi.com/2076-3417/14/19/8868`.

Jarvis K. Bench llm deciders with gym translators. GitHub, 2024. URL `https://github.com/mail-ecnu/Text-Gym-Agents`.

Yash Kant, Arun Ramachandran, Sriram Yenamandra, Igor Gilitschenski, Dhruv Batra, Andrew Szot, and Harsh Agrawal. Housekeep: Tidying virtual households using commonsense reasoning. In European Conference on Computer Vision, pages 355–373. Springer, 2022.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, 32(11):1238–1274, 2013.

Akshay Krishnamurthy, Keegan Harris, Dylan J Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? arXiv preprint arXiv:2403.15371, 2024.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In Reinforcement learning: State-of-the-art, pages 45–73. Springer, 2012.

Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. Advances in Neural Information Processing Systems, 36:43057–43083, 2023.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643, 2020.

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. In 2023 IEEE International Conference on Robotics and Automation (ICRA), pages 9493–9500. IEEE, 2023.

Licong Lin, Yu Bai, and Song Mei. Transformers as decision makers: Provable in-context reinforcement learning via supervised pretraining. arXiv preprint arXiv:2310.08566, 2023.

Shaowei Liu, Zhongzheng Ren, Saurabh Gupta, and Shenlong Wang. Physgen: Rigid-body physics-grounded image-to-video generation. In European Conference on Computer Vision, pages 360–378. Springer, 2024.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. arXiv preprint arXiv:1906.03926, 2019.

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. arXiv preprint arXiv:2310.12931, 2023.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In 2018 IEEE international conference on robotics and automation (ICRA), pages 6292–6299. IEEE, 2018.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. arXiv preprint arXiv:2006.09359, 2020.

Allen Nie, Yi Su, Bo Chang, Jonathan N Lee, Ed H Chi, Quoc V Le, and Minmin Chen. Evolve: Evaluating and optimizing llms for exploration. arXiv preprint arXiv:2410.06238, 2024.

Rohan Pandey. Llamagym: Fine-tune llm agents with online reinforcement learning. GitHub, 2024. URL https://github.com/KhoomeiK/LlamaGym.

Roma Patel and Ellie Pavlick. Mapping language models to grounded conceptual spaces. In International conference on learning representations, 2022.

Arunselvan Ramaswamy and Eyke Hüllermeier. Deep q-learning: Theoretical insights from an asymptotic analysis, 2021. URL https://arxiv.org/abs/2008.10870.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In Machine learning: ECML 2005: 16th European conference on machine learning, Porto, Portugal, October 3-7, 2005. proceedings 16, pages 317–328. Springer, 2005.

Simon Schmitt, Jonathan J Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M Czarnecki, Joel Z Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, et al. Kickstarting deep reinforcement learning. arXiv preprint arXiv:1803.03835, 2018.

Takuma Seno and Michita Imai. d3rlpy: An offline deep reinforcement learning library. Journal of Machine Learning Research, 23(315):1–20, 2022. URL http://jmlr.org/papers/v23/22-0017.html.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. nature, 550(7676):354–359, 2017.

Shrutika Singh, Anton Alyakin, Daniel Alexander Alber, Jaden Stryker, Ai Phuong S Tong, Karl Sangwon, Nicolas Goff, Mathew de la Paz, Miguel Hernandez-Rovira, Ki Yun Park, Eric Claude Leuthardt, and Eric Karl Oermann. It is too many options: Pitfalls of multiple-choice questions in generative ai and medical education, 2025. URL https://arxiv.org/abs/2503.13508.

Yuda Song, Yifei Zhou, Ayush Sekhari, J Andrew Bagnell, Akshay Krishnamurthy, and Wen Sun. Hybrid rl: Using both offline and online data can make rl efficient. arXiv preprint arXiv:2210.06718, 2022.

Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. True knowledge comes from practice: Aligning llms with embodied environments via reinforcement learning. arXiv preprint arXiv:2401.14151, 2024.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. arXiv preprint arXiv:2407.17032, 2024.

Masatoshi Uehara, Xuezhou Zhang, and Wen Sun. Representation learning for online and offline rl in low-rank mdps. arXiv preprint arXiv:2110.04652, 2021.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015. URL https://arxiv.org/abs/1509.06461.

Ruosong Wang, Yifan Wu, Ruslan Salakhutdinov, and Sham Kakade. Instabilities of offline rl with pre-trained neural representation. In International Conference on Machine Learning, pages 10948–10960. PMLR, 2021.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837, 2022.

Tengyang Xie, Nan Jiang, Huan Wang, Caiming Xiong, and Yu Bai. Policy finetuning: Bridging sample-efficient offline and online reinforcement learning. Advances in neural information processing systems, 34:27395–27407, 2021.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2reward: Automated dense reward function generation for reinforcement learning. arXiv preprint arXiv:2309.11489, 2023.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. arXiv preprint arXiv:2412.15115, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629, 2022.

Ruiqi Zhang and Andrea Zanette. Policy finetuning in reinforcement learning via design of experiments using offline data. Advances in Neural Information Processing Systems, 36:59953–59995, 2023.

Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, Proceedings of the 39th International Conference on Machine Learning, volume 162 of Proceedings of Machine Learning Research, pages 27042–27059. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/zheng22c.html`.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. arXiv preprint arXiv:2310.04406, 2023.

A Ablation study

A.1 The environments and implementation details

A.1.1 The environments

We empirically verify our algorithm on some classic RL environments:

- **Cart Pole**: The agent aims to balance a pole on top of a cart by moving left and right. It observes the Cart Position, Cart Velocity, Pole Angle, and Pole Angular Velocity. The reward is one for every step taken before the episode ends, either by having the pole fall over, moving the cart to the edge of the display, or reaching the maximum episode length.
- **Pendulum**: The agent aims to swing up an inverted pendulum by applying torque on its free end. It observes the (x, y) location of the pendulum's free end and its angular velocity. From the location, we calculate the pendulum's angle and the rotating direction to help the LLM, but do not use them in the online phase. The reward is calculated based on the pendulum angle, where the upright location has the highest reward. The episode ends when it reaches the maximum episode length. Note that the action set here is continuous, which can be more challenging for the LLM's policy.
- **Frozen Lake**: The agent aims to move from the top-left to the bottom-right location in a four-by-four grid world. The agent can move up, down, left, and right. It only observes its own location. The reward is zero everywhere except at the goal, where the reward is one. The episode ends either when the agent moves to one of the four "holes" in the grid, reaches the goal, or reaches maximum episode length. We further implement an external environment history to store the rewards received at each visited location, which is necessary for the LLM to solve this task. The environment's history is not used in the online learning phase.
- **Cliff Walking**: The agent aims to move from the bottom-left to the bottom-right location in a four-by-twelve grid world. The agent can move up, down, left, and right. It only observes its own location. The reward is negative one everywhere except negative one hundred at the cliff locations on the bottom of the grid. The episode ends either when the agent reaches the goal or reaches maximum episode length. We also use the environment history for this environment.

- **Represented Pong**: This is the Atari game Pong, but instead of the traditional image observation, we use Anand et al. [2019] to extract the game state information from the RAM state. The agent then observes its own coordination, the ball's, the opponent's, and the score. We also calculate the ball velocity and add it to the observation, since it seems necessary to ensure Markov's property (able to take optimal action with only the current state information). The agent controls the right paddle up and down and competes against the left paddle controlled by the computer by trying to deflect the ball away from your goal and into the opponent's goal. The agent receives a point whenever it scores a goal and loses when the opponent does. The game ends when a player's score reaches twenty-one or the agent reaches the maximum episode length.

- **Mountain Car**: The agent's goal is to move from the bottom of a sinusoidal valley to the top of the right hill as quickly as possible. The agent can strategically accelerate left or right. It only observes its location and velocity. The reward is negative one everywhere except the goal. The episode ends either when the agent reaches the goal or reaches maximum episode length.

A.1.2 Implementation details

We build our code from Pandey [2024], which provides a framework for LLM interacting with OpenAI's gym games with a built-in text description wrapper to turn RL games into something LLM can play. The game descriptions, which are listed in Appendix C, are heavily referenced from K [2024]. The RL training process is using d3rlpy Seno and Imai [2022], with the default hyperparameter choice, with batch-size 256, buffer size 100,000, ε : 0.1, γ : 0.99, target update interval 1,000, and learning rate $5e-5$. We use DDQN van Hasselt et al. [2015] for all tasks with Discrete Action and SAC Haarnoja et al. [2019] for Continuous Action. The LLM was run on two H100 GPUs.

For the LORO algorithm, we collected data to pre-train a policy and then only used online data in the online learning process. In all learning curves, the first $\tau = 10$ episodes in LORO showed the average episode reward using the pure LLM-based policies. Afterwards, LORO significantly outperforms the LLM-based policies and the On-Policy RL baselines.

A.2 Effects of the LLM's capability

Given that the performance of many reasoning tasks increases with the improvement of the LLM's capability through: increasing the model's size, using Supervised Fine-Tuning (SFT), Long Chain-of-Thought (CoT), or some Test-time-scaling methods such as Majority Voting, and Best-of-N. In this section, we want to investigate whether this increase in LLM's reasoning capability also translates to decision making in MDP problems.

From our experiments in Appendix A.3, we see no clear link between an LLM's model size and its performance. Interestingly, we find that a small model size (7B) is more sensitive to edge cases, such as MountainCar (with 3000 pre-training steps in Figure 9), CartPole (with 10 training episodes and 1000 pre-training steps in Figure 7), and FrozenLake (with bad environment history summarization in Figure 26). We also find that the amount of pre-training data in general does not affect the cumulative reward, which was shown in Appendix A.5.

In addition, we notice that the LLM's base policies are only useful using CoT instead of just asking the LLM to make decisions. We also observe that the 0.5B model is not useful, as well as using Majority Voting or Best-of-N without CoT. Hence, in Appendix A.6, we want to see if increasing the LLM's capability using SFT or using an LLM with Long CoT can help. We show that there is no significant difference in using standard CoT compared to using SFT or Long CoT. Understandably, SFT wouldn't be useful, or maybe even be counter-productive, since the amount of data collected for fine-tuning is too small to make a difference (around 500-3000 in our experiments). Hence, we conclude that improvements over LLM's capability does not directly translate to improvement in RL tasks.

A.3 Effects of LLM's model size

In this section, we evaluate the effect of the LLM's model size on the cumulative reward of the policy. We evaluate this with different pre-training data and pre-training steps on six OpenAI Gym environments and show the result in Figure 5, 6, 7, 8, 9, 10. Overall, we observe no clear advantage of using a larger model to improve the decision-making quality of the LORO policy.

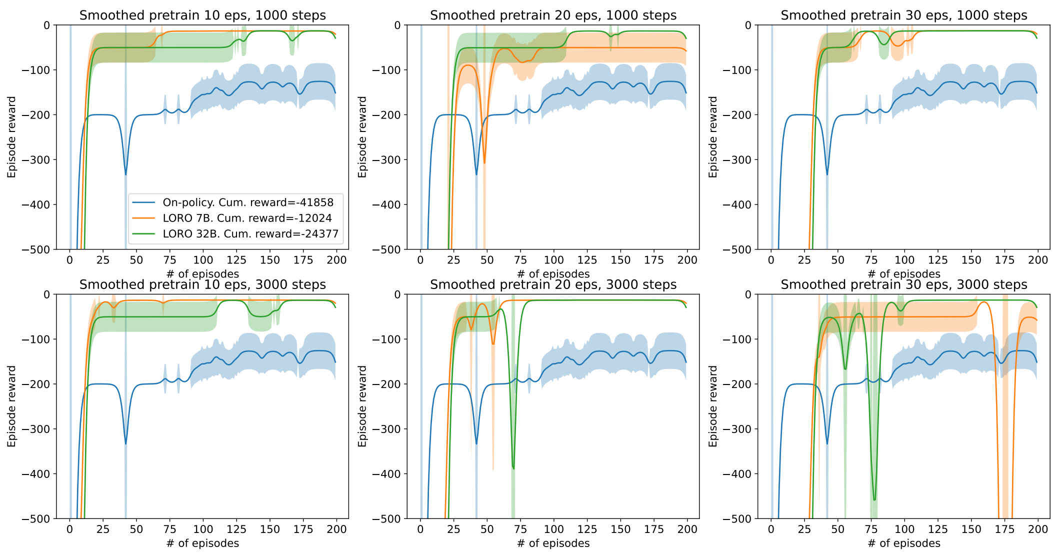

Figure 5: Comparing the effect of different LLMs' model sizes for the CliffWalking environment.

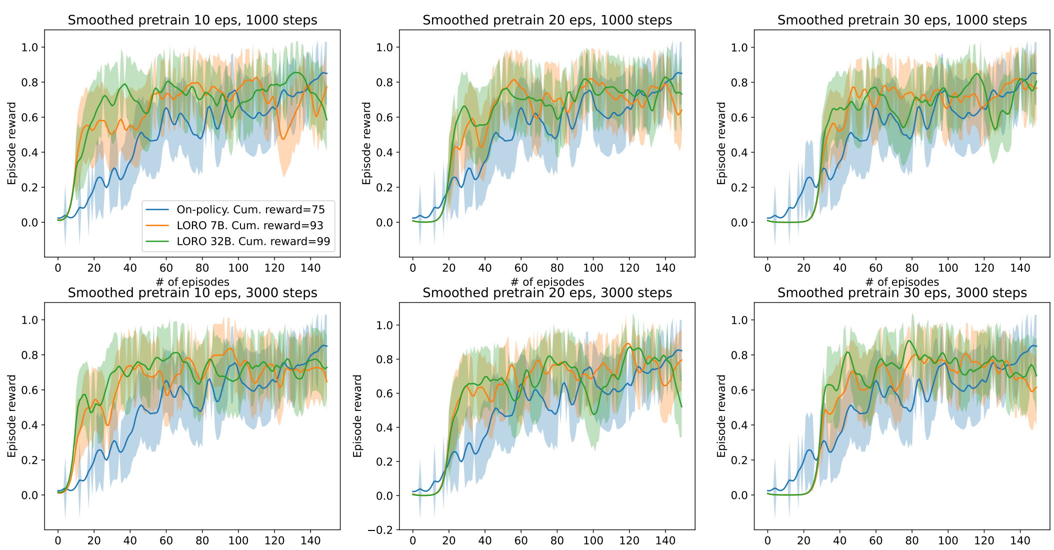

Figure 6: Comparing the effect of different LLMs' model sizes for the FrozenLake environment.

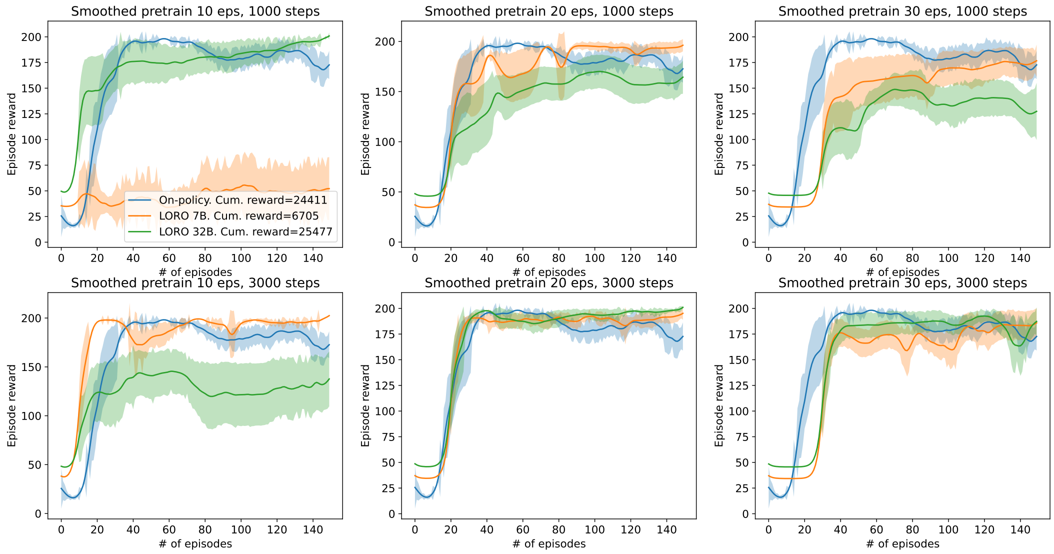

Figure 7: Comparing the effect of different LLMs' model sizes for the CartPole environment.

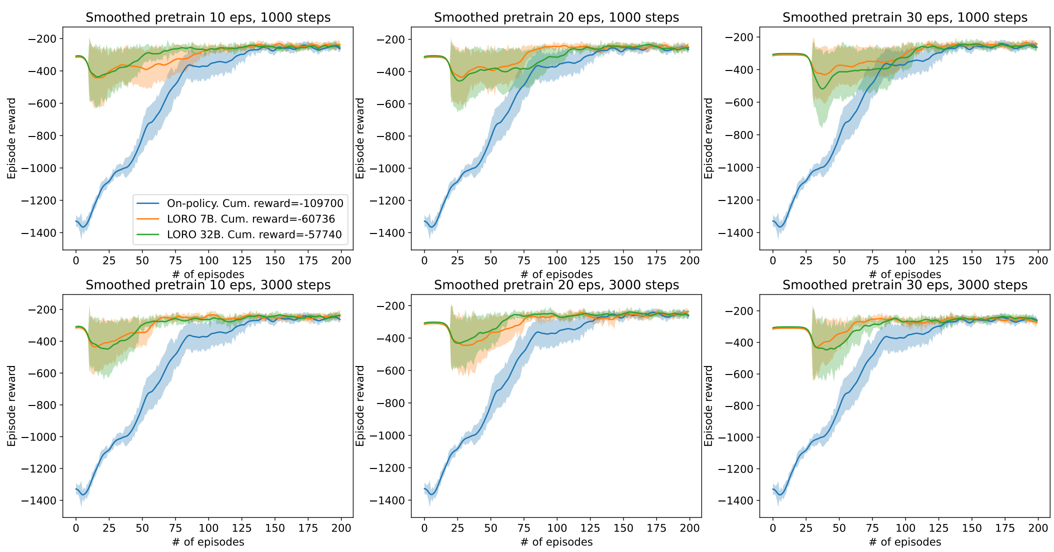

Figure 8: Comparing the effect of different LLMs' model sizes for the Pendulum environment.

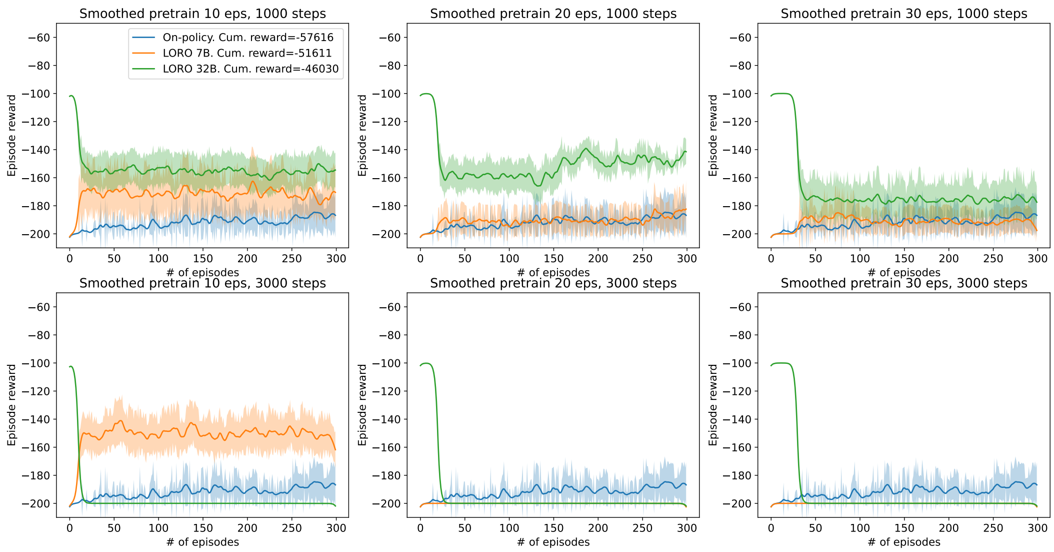

Figure 9: Comparing the effect of different LLMs' model sizes for the MountainCar environment.

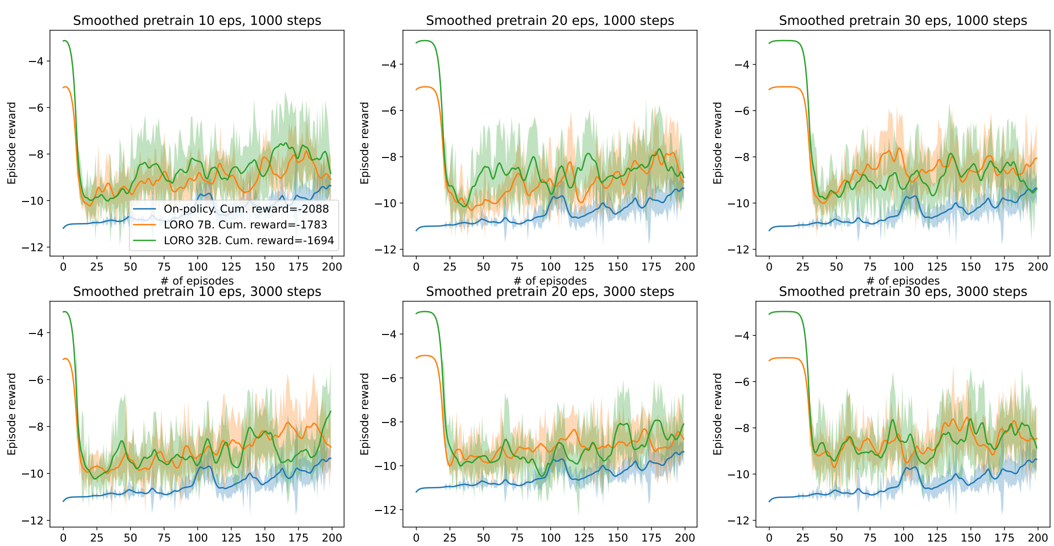

Figure 10: Comparing the effect of different LLMs' model sizes for the Pong environment.

A.4 Effects of the number of pre-training steps

In this section, we evaluate the effect of the number of pre-training steps on the cumulative reward of the policy. We evaluate this with different model sizes and pre-training data on six OpenAI Gym environments and show the result in Figure 11, 12, 13, 14, 15, 16. Overall, we observe no clear advantage of using a higher or lower number of pre-training steps to improve the decision-making quality of the LORO policy.

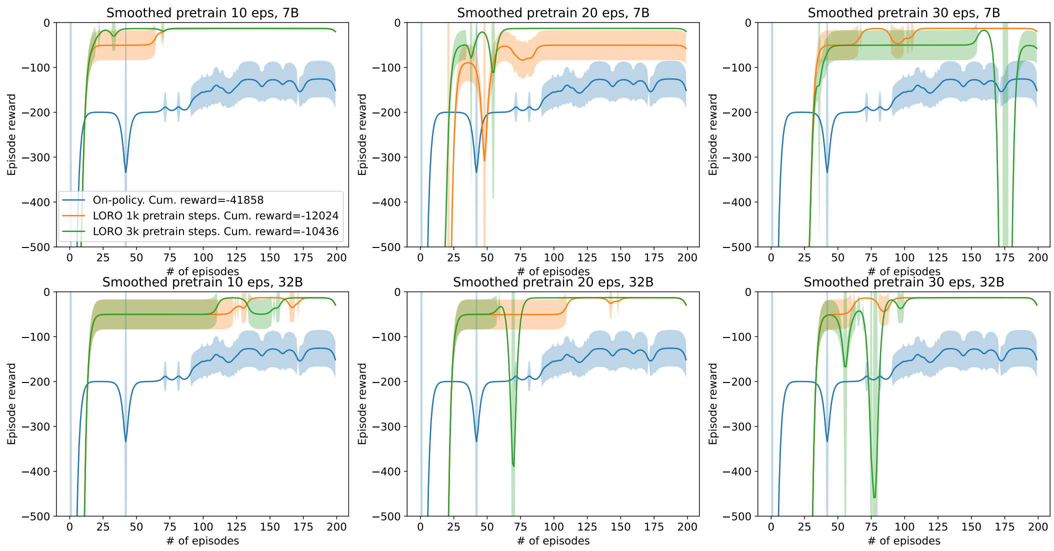

Figure 11: Comparing the effect of different pre-training steps for the CliffWalking environment.

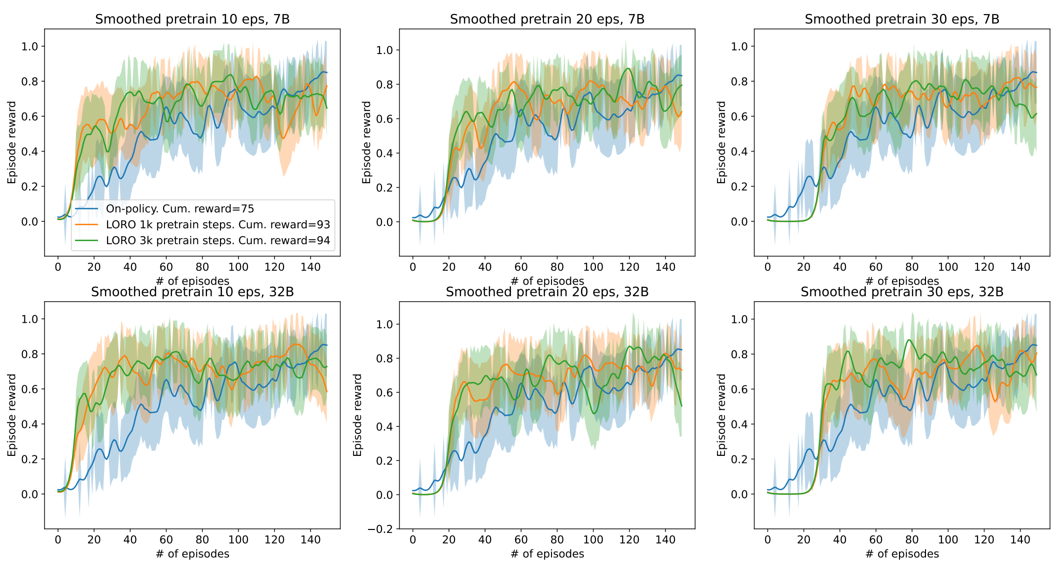

Figure 12: Comparing the effect of different pre-training steps for the FrozenLake environment.

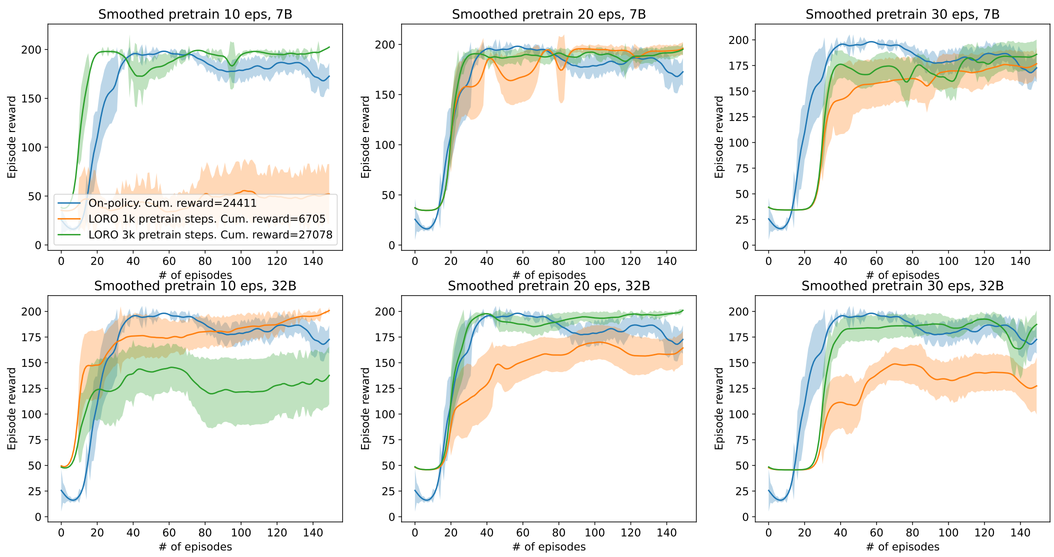

Figure 13: Comparing the effect of different pre-training steps for the CartPole environment.

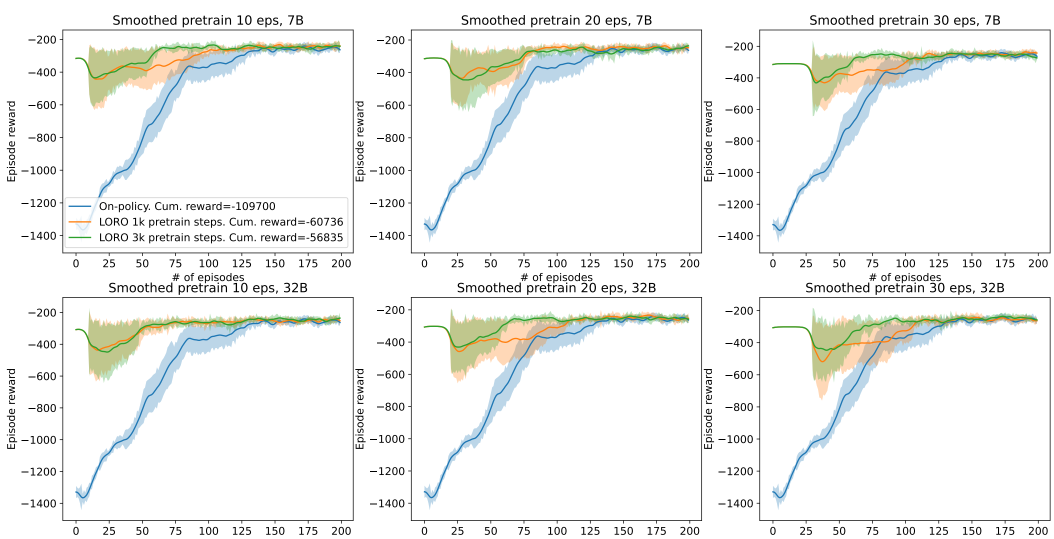

Figure 14: Comparing the effect of different pre-training steps for the Pendulum environment.

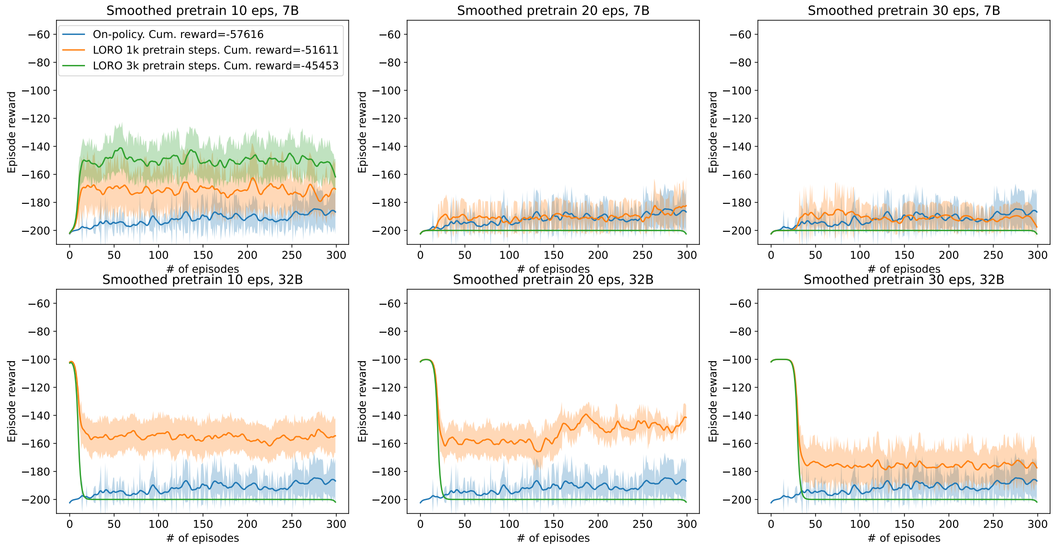

Figure 15: Comparing the effect of different pre-training steps for the MountainCar environment.

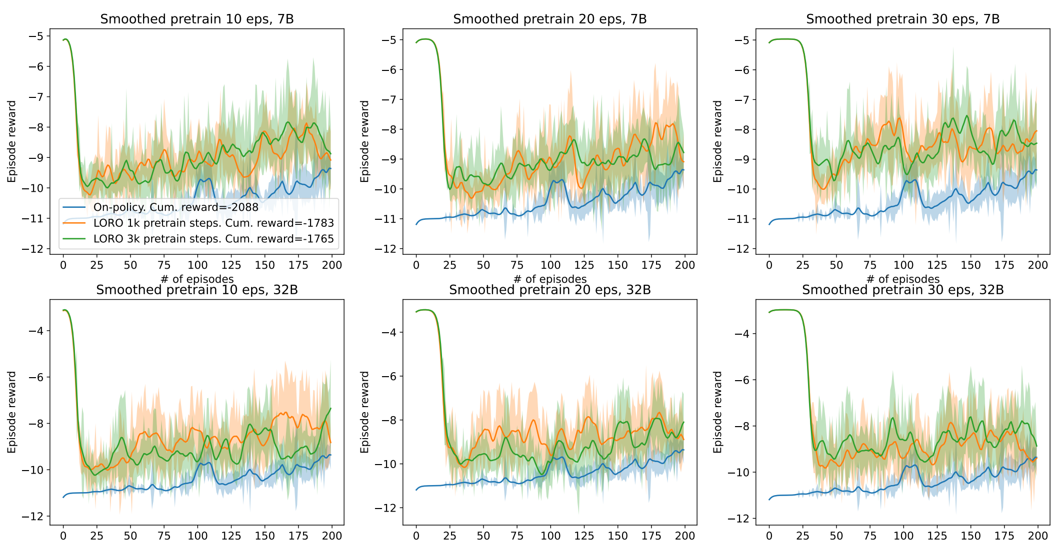

Figure 16: Comparing the effect of different pre-training steps for the Pong environment.

A.5 Effects of the amount of LLM data

In this section, we evaluate the effect of the number of pre-training data on the cumulative reward of the policy. We evaluate this with different model sizes and pre-training steps on six OpenAI Gym environments and show the result in Figure 17, 18, 19, 20, 21, 22. Although there exist some differences in the cumulative reward, all baselines converge to a policy with similar performance in a relatively short amount of time. Hence, we observe no clear advantage of using a higher or lower amount of pre-training data to improve the decision-making quality of the LORO policy.

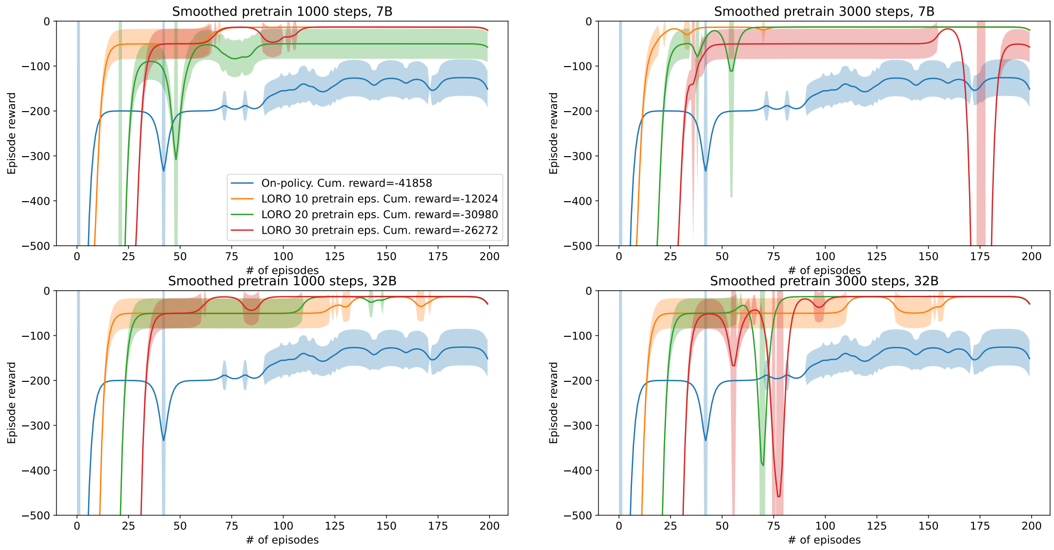

Figure 17: Comparing the effect of different amounts of pre-training data for the CliffWalking environment.

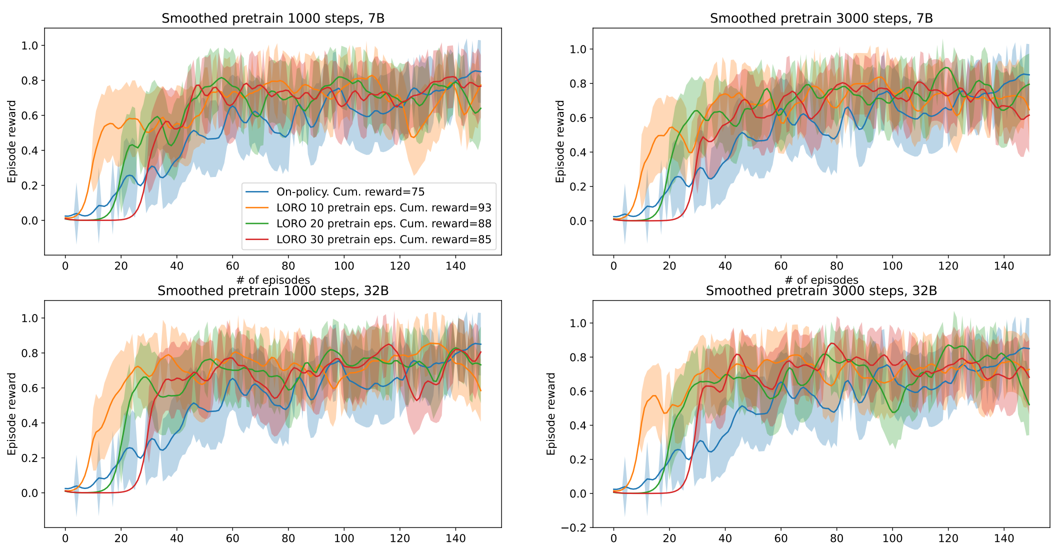

Figure 18: Comparing the effect of different amounts of pre-training data for the FrozenLake environment.

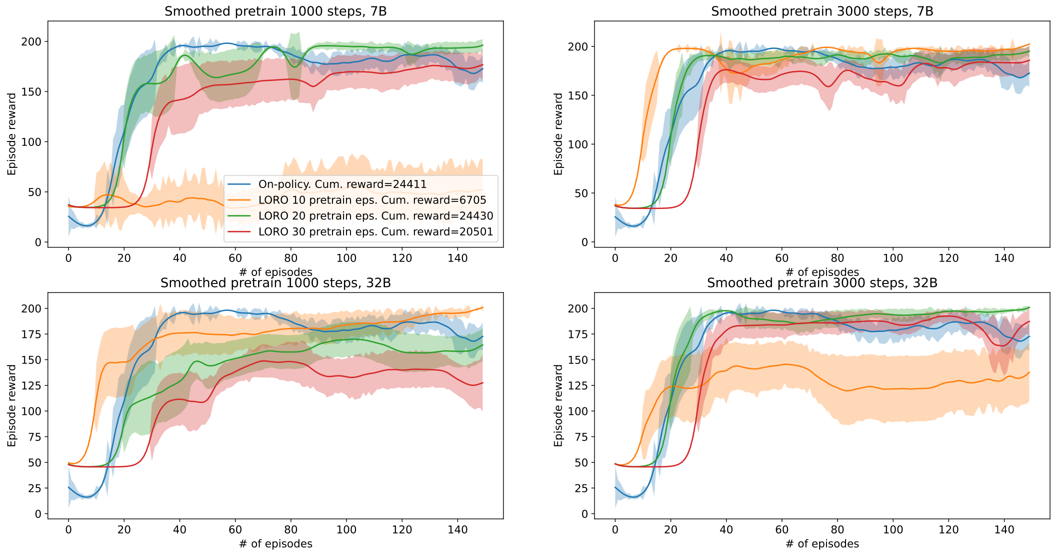

Figure 19: Comparing the effect of different amounts of pre-training data for the CartPole environment.

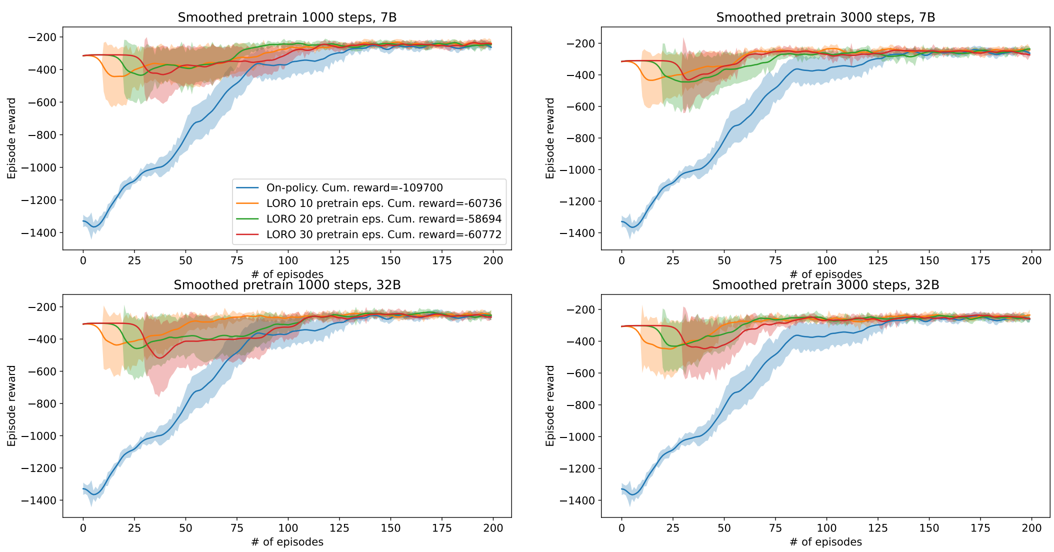

Figure 20: Comparing the effect of different amounts of pre-training data for the Pendulum environment.

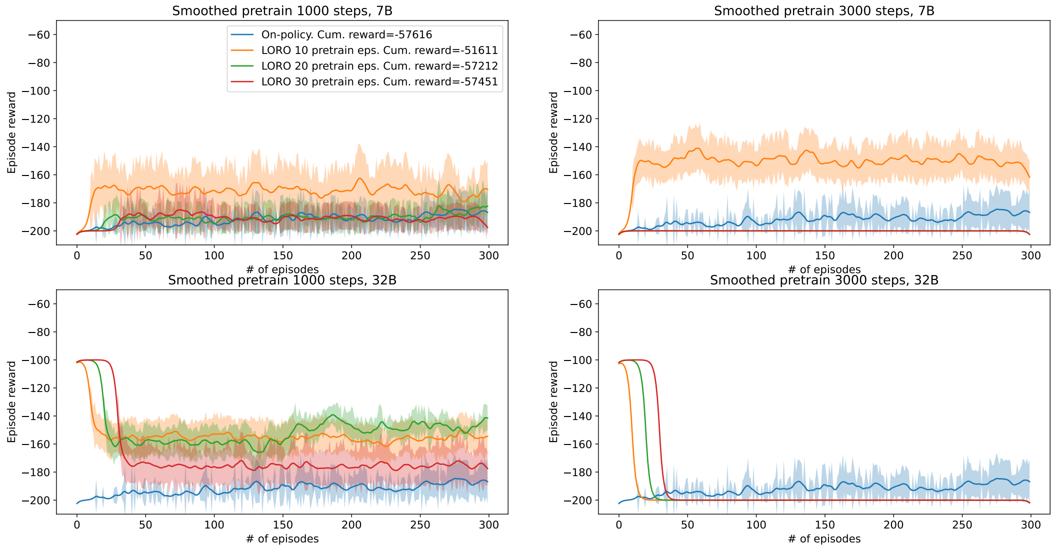

Figure 21: Comparing the effect of different amounts of pre-training data for the MountainCar environment.

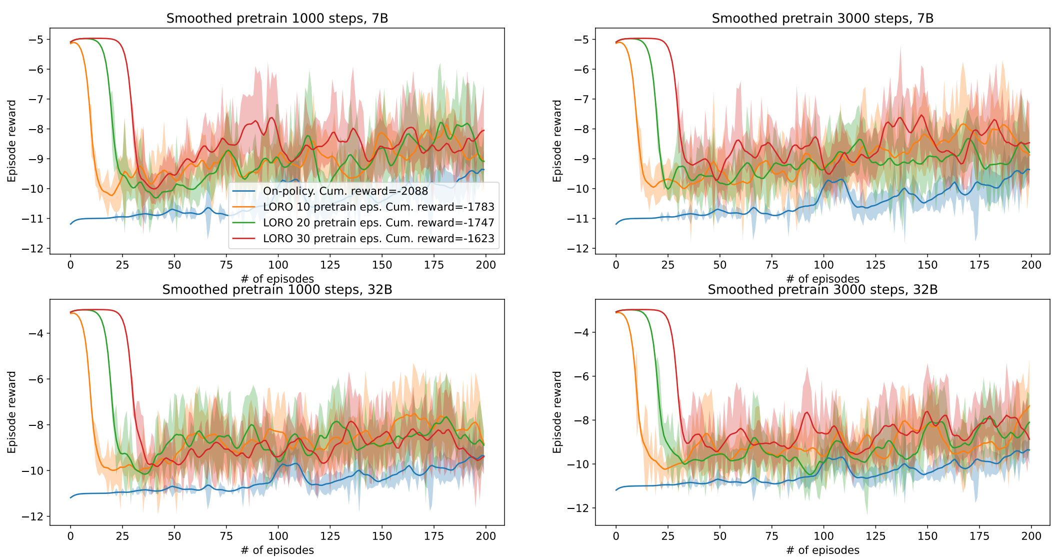

Figure 22: Comparing the effect of different amounts of pre-training data for the Pong environment.

A.6 Effects of SFT and Long CoT

In this section, we evaluate the effect of SFT and Long CoT on the cumulative reward of the policy. We evaluate this with different pre-training data and pre-training steps on three OpenAI Gym environments and show the result in Figure 23, 24, 25. Overall, we observe no clear advantage of using SFT and Long CoT over vanilla CoT to improve the decision-making quality of the LORO policy.

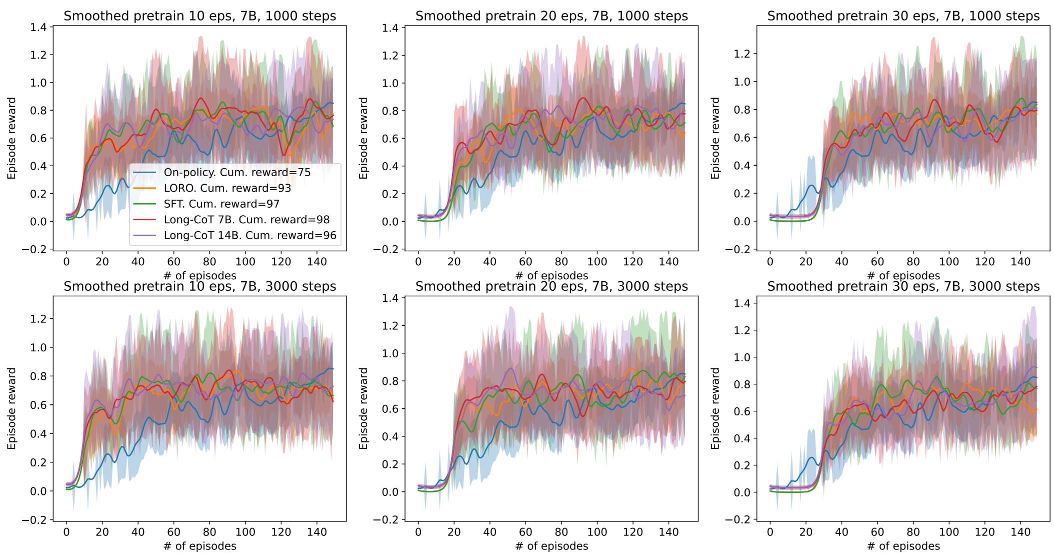

Figure 23: Comparing the effect of Long Chain-of-Thought and Supervised-Fine-Tuning for the FrozenLake environment.

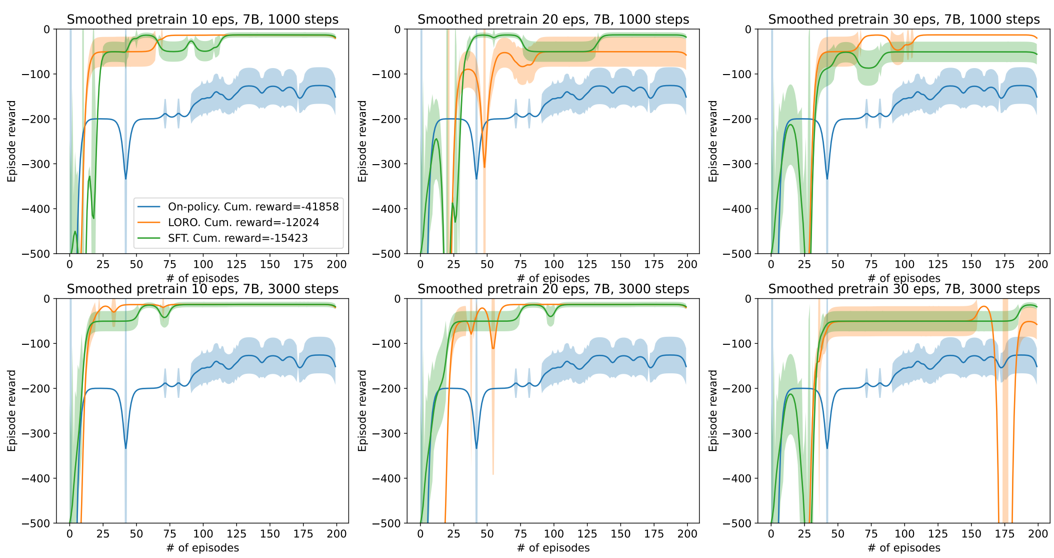

Figure 24: The effect of Supervised-Fine-Tuning for the CliffWalking environment.

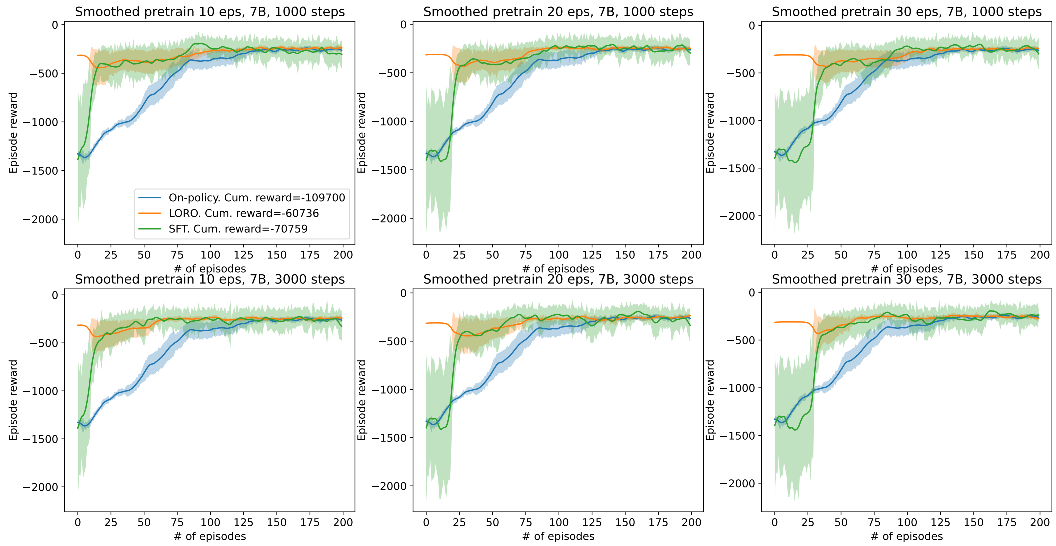

Figure 25: The effect of Supervised Fine-Tuning for the Pendulum environment.

A.7 Effects of the history summary

For the experiments above, we use an efficient environment history such as “The holes are in locations: X, Y, Z. You receive zero reward at locations: A, B, C, D”.

For the experiment in Figure 26, we concatenate the observations of each state to the LLM’s prompt, with a limited history length: "You visit location X and receive zero reward. You visit location Y and receive one reward. You visit ...".

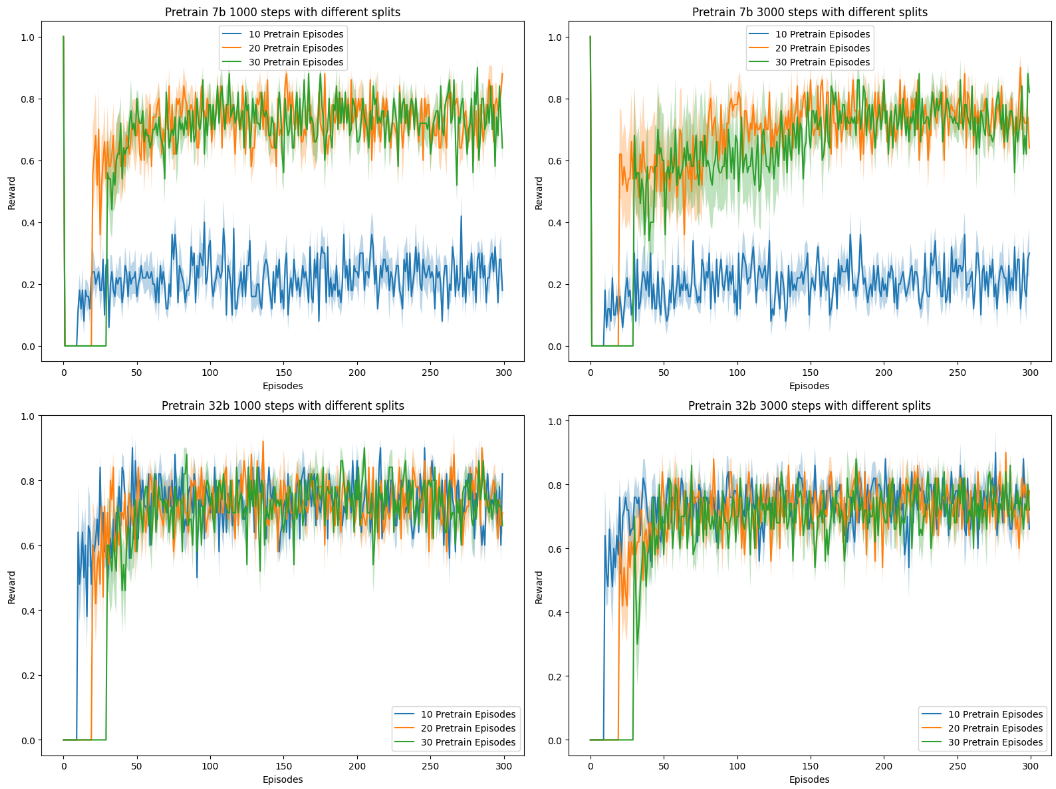

Figure 26: FrozenLake with ineffective environment history.

B LLM setup

We designed the prompt to choose an action from a list of integers starting from one, since we observed that LLM is more biased toward action zero. After the LLM chooses an action, we extract it by getting the last number returned by the LLM. This design was inherited from Pandey [2024], which can be improved since we observe a number of extraction failures from our experiments.

We observe that the vanilla design of LLM, where we ask it only to return the chosen action, performs poorly. Similarly, we implemented and tested the Majority Voting and Best-of-N test-time-scaling methods, but they both perform poorly without CoT.

For all experiments, we limit the generating token to be less than 2000 , top-p 0.6, top-k 0, temperature 0.9. In the SFT experiment, the LLM was trained with LORA Hu et al. [2021] with rank 8, alpha 16, dropout 0.05, batch size 1, and using 8-bit quantization.

C Prompts

This section provides a collection of Chain-of-Thought (CoT) prompting examples used across multiple decision-making environments. Each task is presented with a detailed system prompt describing the environment dynamics and action space, followed by a user query that reflects a specific game state. The responses are generated by `Qwen2.5-7B-Instruct`, illustrating its step-by-step reasoning capabilities.

C.1 Chain-of-Though Prompts

C.1.1 Task 1: Cliff Walking

A Sample of Cliff Walking

► **System:**
You are an expert-level game player. Cliff walking is a task in which you control a player navigating a '4x12' grid world. The ('x', 'y') coordinate indicates the position at row 'x' and column 'y'. The player needs to find a goal location while avoiding cliffs . The player can choose from 4 actions: move up, move right, move down, or move left. If the player takes an action at ('x', 'y'), he tries to move to ('a', 'b'). Rules:

- 1. If ('a', 'b') is a cliff, the player incurs a large penalty of -100, and is reset to the starting position.
- 2. If ('a', 'b') is safe or towards the grid boundary, results in a small penalty of -1. If ('a', 'b') is outside the grid's boundaries, it does not change position but still receive the -1 penalty.
- 3. The game ends when the ('a', 'b') is the goal or 200 actions are performed. The goal is to navigate from the starting point to an target , while avoiding the cliff, in as few steps as possible.

Type a number to indicate the action. For current position ('x', 'y'), the action means the player try to step into the next position. Type '1' to move up, which means trying to step into ('x-1', 'y'), '2' to move right, which means ('x', 'y+1'), '3' to move down, which means ('x+1', 'y'), or '4' to move left, which means ('x', 'y-1'). Ensure you only provide the action number from the valid action list, i.e., [1, 2, 3, 4]. Environment history: Cliff: Reward -100 at locations: (3, 0). Reward -1 at locations: (2, 0), (3, 0), (1, 0). Previous location: (1, 0), previous action: 1, previous reward: -1. Return the action at the end of your answer without the target's location.

► **User:**
You are at location (2, 0) in the grid world.
Think step by step.

► **Qwen2.5-7B-Instruct:**
1

C.1.2 Task 2: CartPole

A Sample of CartPole

► **System:**
You are an expert-level game player. In the CartPole game, you control a cart that moves along a horizontal track. There is a pole standing upright on the cart. The goal of the game is to keep the pole balanced upright by moving the cart left or right. The game ends if the pole tilts too far from the vertical position or if the cart moves too far from the center of the track. The longer you can keep the pole balanced, the higher your score.Note that when the Cart Position is out of the (-2.4, 2.4) zone or the Pole Angle is out of the zone (-.2095, .2095), the round ends and the game is lost. The goal is to keep the pole balanced upright for as long as possible.
Type '1' to push the cart to the left or '2' to push the cart to the right. Ensure you only provide the action number from the valid action list, i.e., [1, 2].

► **User:**
The cart is positioned at 0.006, with a velocity of 0.04 towards the right. The pole is tilted at 0.02 radians, rotating at 0.02 radians per second towards the right.
Think step by step.

► **Qwen2.5-7B-Instruct:**
Given the current state of the CartPole game:

- Cart Position: 0.006 (slightly to the right of the center)

- Cart Velocity: 0.04 (moving slightly to the right)
- Pole Angle: 0.02 radians (slightly tilted to the right)
- Pole Angular Velocity: 0.02 radians/second (tilting to the right)

Let's analyze the situation step by step:

1. **Cart Position**: The cart is very close to the center, at 0.006. This is within the safe range, so we don't need to move it significantly.
2. **Cart Velocity**: The cart is moving slightly to the right, which is good because it helps counteract the tilt of the pole.
3. **Pole Angle**: The pole is tilted 0.02 radians to the right. This is a small tilt, but it indicates that the pole is starting to lean. We need to act quickly to prevent it from falling over.
4. **Pole Angular Velocity**: The pole is rotating at 0.02 radians/second to the right, which means it is accelerating its tilt. This suggests that the cart's movement is not sufficient to stabilize the pole.

Given these observations, we should try to slow down the pole's tilt by moving the cart to the left. This will help counteract the rightward tilt of the pole.

Action: 1 (Push the cart to the left)

This action will help to stabilize the pole by moving the center of mass back towards the center, which should reduce the angular velocity of the pole.

C.1.3 Task 3: Mountain Car

A Sample of Mauntain Car

► **System:**
You are an expert-level game player. In the Mountain Car game, you control a car placed stochastically at the bottom of a sinusoidal valley. The only possible actions are the accelerations that can be applied to the car in either direction. The goal of the game is to strategically accelerate the car to reach the goal state on top of the right hill as quickly as possible. The episode ends if either the car reaches the goal position on top of the right hill or the length of the episode is 200. The goal is to reach the flag placed on top of the right hill as quickly as possible.
Type '1' to accelerate to the left, '2' to not accelerate, or '3' to accelerate to the right.Ensure you only provide the action number from the valid action list, i.e., [1, 2, 3].

► **User:**
The car is positioned at -0.541, with a velocity of 0.000 towards the left.
Think step by step.

► **Qwen2.5-7B-Instruct:**
Given the car's position at -0.541 and a velocity of 0.000 towards the left, the car is slightly to the left of the center of the valley. Since the car is not moving yet and is close to the middle, accelerating to the right (action 3) might help gain some speed towards the right hill without overshooting too much.

Action: 3

C.1.4 Task 4: FrozenLake

A Sample of FrozenLake (CoT)

► **System:**
You are an expert-level game player. In the FrozenLake game, the player starts at the start position of the grid and tries to reach the goal position . There are holes which the player must avoid. The frozen lake is slippery, meaning that the player might not always move in the intended direction. The game ends when the player reaches the goal or falls into a hole.

The goal is to navigate across the frozen lake and reach the goal position without falling into any holes. For current position ('x', 'y'), the action means the player try to step into the next position. The possible actions are:

- 1: Move left, which means ('x', 'y-1'),
- 2: Move down, which means ('x+1', 'y'),
- 3: Move right, which means ('x', 'y+1'),
- 4: Move up, which means trying to step into ('x-1', 'y').

Ensure you only provide the action number from the valid action list, i.e., [1, 2, 3, 4]. Do not return the target's coordination. Environment history: Reward 0.0 at locations: (0, 0). Previous location: (0, 0), previous action: 1, previous reward: 0.0. Return the action at the end of your answer without the target's location.

► **User:**
You are at row 0, column 0.
Think step by step.

► **Qwen2.5-7B-Instruct:**
Since the current position is (0, 0) and the previous action was moving left (which is not possible as it would take you out of bounds), the only feasible action to consider is moving down to (0, 1). Therefore, the action is: **2**.

A Sample of FrozenLake (Long CoT)

► **System:**
You are an expert-level game player. In the FrozenLake game, the player starts at the start position of the grid and tries to reach the goal position . There are holes which the player must avoid. The frozen lake is slippery, meaning that the player might not always move in the intended direction. The game ends when the player reaches the goal or falls into a hole. The goal is to navigate across the frozen lake and reach the goal position without falling into any holes. For current position ('x', 'y'), the action means the player try to step into the next position. The possible actions are:

- 1: Move left, which means ('x', 'y-1'),
- 2: Move down, which means ('x+1', 'y'),
- 3: Move right, which means ('x', 'y+1'),
- 4: Move up, which means trying to step into ('x-1', 'y').

Ensure you only provide the action number from the valid action list, i.e., [1, 2, 3, 4]. Do not return the target's coordination. Environment history: Reward 0.0 at locations: (0, 0). Previous location: (0, 0), previous action: 1, previous reward: 0.0. Return the action at the end of your answer without the target's location.

► **User:**
The current position of the You are at row 0, column 0.
Think step by step.

► **DeepSeek-R1-Distill-Qwen-7B:**

Okay, so I'm trying to figure out where to move in this FrozenLake game. I'm at position (0,0), which is the top-left corner of the grid. The goal is to reach the goal position, which I think is at the bottom-right corner, maybe (4,4) since it's a 5x5 grid. But I'm not entirely sure, so I'll assume that's the case.
First, I need to consider the possible actions. The actions are numbered 1 to 4, corresponding to moving left, down, right, and up respectively. Each action has a direction, but because the ice is slippery, I might not end up where I intended. So, I have to plan carefully to avoid falling into holes.

Let me think about each action:

- Action 1: Move left. But wait, I'm already at column 0, which is the leftmost column. So moving left from here would take me off the grid, which is probably a hole. So I shouldn't choose action 1.

- Action 2: Move down. That would take me to (1,0). I need to check if that's a hole. From the grid, I remember that some cells are safe and some are holes. Let me visualize the grid:
  - Row 0: S F F F F
  - Row 1: F F H F F
  - Row 2: F H F F F
  - Row 3: F F F H F
  - Row 4: F F F F G

  Wait, I'm not sure if that's accurate, but I think row 1, column 2 is a hole (H), and row 2, column 1 is another hole. So moving down to (1,0) is safe because row 1, column 0 is F (safe). So action 2 seems safe.
- Action 3: Move right. That would take me to (0,1). I think that's safe too because row 0, column 1 is F.
- Action 4: Move up. But I'm at row 0, so moving up would take me off the grid, which is a hole. So action 4 is out.

So, from (0,0), the valid actions are 2 and 3. I need to choose between them. Let's think about the long-term implications.
If I go down to (1,0), then from there, possible actions are:

- Move left: off grid (hole)
- Move down: (2,0) which is safe
- Move right: (1,1) which is safe
- Move up: back to (0,0)

If I go right to (0,1), then from there:

- Move left: back to (0,0)
- Move down: (1,1) which is safe
- Move right: (0,2) which is safe
- Move up: off grid (hole)

Hmm, both paths seem viable. Maybe I should go right to (0,1) because moving down from there might lead me closer to the goal without getting stuck. Alternatively, going down might allow me to move right or left without immediate issues.

Wait, but if I go down to (1,0), then moving right to (1,1) is safe, and from there, moving down again to (2,1) is a hole because row 2, column 1 is H. Oh no, that's a problem. So from (1,0), if I move right to (1,1), then down to (2,1) is a hole. So that's bad.
Alternatively, from (1,0), if I move down to (2,0), which is safe, then from there, moving right to (2,1) is a hole. So again, that's a problem.

Wait, maybe I'm misremembering the grid. Let me try to recall the layout. I think row 1, column 2 is H, and row 2, column 1 is H. So, from (1,0), moving right goes to (1,1), which is safe, but then moving down from there would go to (2,1), which is H. So that's a problem.
Alternatively, from (1,0), moving down to (2,0) is safe, but then moving right to (2,1) is H. So either way, moving down first might lead to a hole in the next step.
So maybe going right to (0,1) is better. From (0,1), moving down to (1,1) is safe, and then from there, moving right to (1,2) is H, which is bad. Or moving right to (0,2), which is safe, then from there, moving down to (1,2) is H. So again, same issue.

Wait, maybe I'm overcomplicating this. Perhaps I should try to find a path that avoids the holes. Let's think about possible paths.
From (0,0), go right to (0,1). From there, go down to (1,1). From (1,1), go right to (1,2), but that's H. So that's bad. Alternatively, from (1,1), go right to (1,2) is H, so maybe go down to (2,1), which is H. So that's no good.
Alternatively, from (0,1), go down to (1,1), then from there, go right to (1,2) is H, or left to (1,0), which is back where I came from. Hmm.

Wait, maybe from (0,1), go right to (0,2). From (0,2), go down to (1,2) is H. So that's bad.

Alternatively, from (0,1), go down to (1,1), then from there, go right to (1,2) is H, or left to (1,0), which is safe. So from (1,1), moving left to (1,0) is safe, but then from (1,0), moving down to (2,0) is safe, and from there, moving right to (2,1) is H. So that's a problem.
This is getting complicated. Maybe I should try a different approach. Let's think about the possible paths step by step.
From (0,0), options are 2 (down) and 3 (right).
If I go down to (1,0):

- From (1,0), possible actions: down to (2,0), right to (1,1)
- If I go down to (2,0):
  - From (2,0), possible actions: right to (2,1) is H, so that's bad. So maybe go right to (2,1) is H, so can't go there. So from (2,0), the only safe move is right to (2,1) is H, so maybe move up to (1,0), but that's where I came from. Or move right to (2,1) is H, so that's not allowed. So from (2,0), the only safe move is to go right to (2,1), but that's H, so that's bad. Alternatively, move left to (2,-1), which is off grid, so that's a hole. So from (2,0), I can't move anywhere without falling into a hole. So that's a dead end.

So going down to (1,0) leads to a dead end because from (2,0), I can't move anywhere safely.

So, maybe I should go right to (0,1) instead.
From (0,1):

- Possible actions: down to (1,1), right to (0,2)
- If I go down to (1,1):
  - From (1,1), possible actions: down to (2,1) is H, right to (1,2) is H, left to (1,0), up to (0,1)
  - So, from (1,1), the only safe move is left to (1,0) or up to (0,1). If I go left to (1,0), then from there, I can go down to (2,0), but from (2,0), as before, it's a dead end. Alternatively, from (1,1), go up to (0,1), which is where I came from, so that's not helpful.
- If I go right to (0,2):
  - From (0,2), possible actions: down to (1,2) is H, right to (0,3), left to (0,1)
  - So, from (0,2), the safe moves are right to (0,3) or left to (0,1). If I go right to (0,3), then from there, I can go down to (1,3), which is safe, and so on.

So, from (0,1), going right to (0,2) seems better because it allows me to move further right without hitting a hole immediately.

So, let's try that path:
Action: 1

C.1.5 Task 5: Pong

A Sample of Pong

► **System:**
You are an expert-level game player. In the Pong game, you play the ball with your opponent, each player rallys the ball by moving the paddles on the playfield. Paddles move only vertically on the playfield. A player scores one point when the opponent hits the ball out of bounds or misses a hit. The first player to score 21 points wins the game. The number of frameskip is set to 4. The goal is to knock out your opponent. Type 1 for NOOP (no operation), 3 to move up, 4 to move down. Ensure you only provide the action number from the valid action list, i.e., [1, 3, 4].

► **User:**
The origin (0,0) is in the top left corner. You are at position (188, 78), your opponent is at position (64, 8), the ball is at (0, 0), the ball velocity is (0, -60). Your opponent's score is 0, your score is 0.
Think step by step.

► **Qwen2.5-7B-Instruct:**
Given the current state of the game:

- Your paddle is at position (188, 78).
- Your opponent's paddle is at position (64, 8).
- The ball is at position (0, 0) and is moving upwards with a velocity of (0, -60).

Step-by-step reasoning:

- The ball is at the top of the playfield and is moving downwards. It will be near the middle of the playfield soon.
- With a frameskip of 4, the opponent will not react until the ball has moved significantly downward.
- Since the ball is about to come towards you, you should prepare to move your paddle to intercept the ball.

Action Plan:

- Move your paddle up to intercept the ball as it comes towards you. This ensures that you can return the ball effectively.

Action: 3 (move up)
By moving up, you position yourself to intercept the ball as it comes towards the center of the playfield.

C.1.6 Task 6: Pendulum

A Sample of Pendulum

► **System:**
You are an expert-level game player. In the Pendulum game, you control a pendulum attached to a fixed pivot point. The goal is to apply torques to swing the pendulum upright and keep it balanced. The game ends if the pendulum cannot be stabilized within the given time limit. The closer the pendulum is to the upright position, the higher your score. The goal is to swing the pendulum upright and keep it balanced. Provide a torque value (e.g., a float between -2.0 and 2.0) to control the pendulum's movement. Return the torque value enclosed in < and >, e.g., <1.5>.

► **User:**
The pendulum is at an angle of -2.690 radians from the vertical (zero when upright), rotating at 0.34 radians per second in the clockwise direction.
Think step by step.

► **Qwen2.5-7B-Instruct:**
To determine the appropriate torque to apply, we need to consider the current state of the pendulum: its angle and angular velocity.

- **Angle Analysis**: The pendulum is currently at an angle of -2.690 radians from the vertical. This means it is hanging to the left of the vertical position. To bring it back towards the vertical, we need to apply a positive torque (since positive torque will tend to rotate the pendulum clockwise, which is the direction needed to move it towards the vertical).
- **Angular Velocity Analysis**: The pendulum is rotating at 0.34 radians per second in the clockwise direction. Since the pendulum is already moving in the direction we want (towards the vertical), we should continue to apply a positive torque to help it swing more quickly towards the vertical position.

Given that the pendulum is already moving in the correct direction but not fast enough, applying a moderate positive torque should help it swing faster and reach the vertical position more efficiently.
A good starting torque value could be around 1.0, as this provides a reasonable amount of force without being too aggressive and risking overshooting the vertical position.
Therefore, the torque value I suggest is: **<1.0>**