Extended Abstract

Motivation Legged robots have the potential to traverse disaster zones or extraterrestrial environments but robust control algorithms that generalize are needed. Quadruped parkour is a useful context to study this problem. Current methods like optimization may struggle with runtime or modeling complex environment interaction like stepping on a large obstacle Kim et al. (2019). Reinforcement learning (RL) approaches may struggle to add new skills once trained Hoeller et al. (2024) or they may rely on certain obstacles during training and risk failing to generalize to new unseen obstacles Cheng et al. (2024). We focus on the generalization of reinforcement learning approaches and test a baseline method from recent literature in comparison to our method using a mixture of experts on unseen (to the training networks) obstacles.

Method To address the generalization problem, we propose using a mixture of experts (MoE) as the actor in an actor-critic Proximal Policy Optimization (PPO) RL algorithm. Specifically it relies on a gating network to produce weights for the experts as a discrete probability distribution using some suggestions from Shazeer et al. (2017). This is used as an input into a mixture of gaussians as the selecting distribution. The cohort of experts, also MLPs, produce the means for the individual gaussian probability distributions, std deviations are learned parameters. A term to encourage diversity of experts is added to the objective. To the best of our knowledge this has not been done in a parkour setting with vision, though it has been done in a pre-print examining a parkour setting without vision Huang et al. (2025). Additionally, experimental results on novel obstacles that were unseen in training are also shown.

Implementation We implement a python library for RL learning in this parkour context, building on pytorch as well as the Legged Gym environment from Cheng et al. (2024) that leverages Nvidia Isaac Gym. Our library includes a PPO implementation, MoE Actor, and a new obstacle, among other things. We use docker on a g4dn.xlarge Amazon EC2 instance. We also train locally on an Nvidia RTX 2060 graphics processing unit (GPU). We use 2048 environments in parallel and 24 timesteps per collection step. We use this to train and take 8 epochs using the collection data, separating it into mini batches for each epoch. This is one learning step. We then repeat. 6k steps is 300 million individual steps if using 2048 parallel environments.

Results We track the mean terrain level, given the environment makes tasks more difficult, and mean reward. These two we report at 6k training steps. We have 4 policies including: (1) the baseline approach from Cheng et al. (2024) with 6144 parallel environments that has a mean terrain level of 5.97 and a mean reward of 15.61. (2) A modified baseline (to simplify their PPO to be closer to ours) with 2048 environments (what our algorithms use) in parallel has a mean terrain level of 6.05 and a mean reward of 9.91. (3) The MoE approach using 1 big mini-batch, 5 experts, and no sparsity has a mean terrain level of 4.90 and a mean reward of 10.91. (4) The MoE approach using 4 mini-batches, 5 experts, and the top 2 experts for weights (using sparsity) has a mean terrain level of 1.10 and a mean reward of 4.40. We test the four algorithms on the new obstacle, that is similar to a rocky field in nature. We run 10 robots in parallel on the obstacle and track what percent succeed in reaching the goal that requires navigating across the field, in 75 seconds. (1) trained on 14.5k steps was 0% succesful. (2) trained on 13k steps was 0% succesful. (3) was 30% succesful, with 6k steps. (4) with 7k steps was 0% succesful.

Discussion We show that the MoE approach is slower to train, it gets less terrain level increases than other approaches. However, this may be expected because it trains many more networks than the baselines. It also requires more computations to get multiple expert distributions and blend them to produce actions. However, the MoE approach without sparsity was the only approach that had some success in the unseen obstacle trial, showing a step towards generalization and this at a lower number of training steps by a factor of 2 to the baseline.

Conclusion The study on MoE approaches shows promise to improve generalization. However, these results should be taken with caution as they were based on phase 1 training leveraging privileged data like scandots and way-points that a real robot would not have access to. Repeating with phase 2 data is recommended. Additionally, the MoE approaches were tested before the domain randomization compaign at 8k steps (due to some spot instance restarts). This should be repeated.

Quadruped Parkour–Mixture of Experts with Visual Input to Enable Generalization

Michael Ziegltrum
Department of Computer Science
Stanford University
zieglm@stanford.edu,ziegltrum@gmail.com

Abstract

Quadruped parkour is useful to enable robots to traverse rugged environments like disaster zones. Existing RL approaches fail to generalize to new obstacles and don't allow adding new skills easily. We propose an approach using visual input and a mixture of experts to enable generalization. We train a policy using privileged data in simulation and show that it achieves 81% of the baseline's terrain level using far fewer training steps. It also has a success rate of 30% on a novel obstacle that is dissimilar to obstacles in the training environment, compared to 0% of the baseline. This comes at the cost of slower training and higher computation requirements, given multiple experts must be leveraged.

1 Introduction

In July 1969, astronauts Armstrong, Collins, and Aldrin were the first of humanity to set foot on the moon. Armstrong intoned, "that's one small step for [a] man, one giant leap for mankind." It is perhaps no coincidence that Armstrong chose the word step, nor that humans evolved as bipeds. A large part of the natural world, in addition to most of the man-made world, is optimized for legged creatures. Legged robots are therefore an attractive robotic design though advanced control schemes are needed to support this locomotion. The objective of this project is to learn a policy to control a quadruped to smoothly navigate over rugged environments, and show some success in navigating obstacles not seen in training. We place the emphasis on traversing novel obstacles. This is important because for legged robots to be useful in mine fields or disaster zones they may need to deal with obstacles unseen in simulation.

2 Related Work

Multiple strategies have been proposed for dynamic quadruped locomotion on challenging terrain including model predictive control (MPC), trajectory optimization (TO), imitation learning, and model-free reinforcement learning (RL).

MPC strategies leverage dynamics models like the classic single rigid-body dynamics (SRBD or potato model) and may linearize it to apply linear MPC or they may alternatively use more modern dynamics like whole body control (WBC) and non-linear MPC, though WBC and non-linear MPC come at a computation and therefore time cost Katayama et al. (2023). This makes it challenging for highly dynamic movements though Kim et al. (2019) used MPC to find an optimal reaction force and then WBC to compute joint torque, position and velocity commands and achieved highly dynamic movement like running at a speed of 3.7m/s, though they do not show interaction with obstacles in the environment. This means it doesn't meet the objective of traversing novel obstacles.

Trajectory optimization (TO) is another approach employed in Bjelonic et al. (2020), where the authors use an online TO to optimize wheel and base trajectories for a quadruped with wheels as feet and show success in rough terrain like navigating steps up to 20% of the robot's leg length, though they have no automated switch between pure driving and hybrid walking. This means it doesn't meet the objective of smoothly navigating.

Learning based approaches are another strategy including imitation learning like that of Peng et al. (2020) who show dynamic hops and turns by imitating motion reference data, though they were not able to show highly dynamic movement like large jumps or runs. Model-free RL strategies like Rudin et al. (2022) leverage deep reinforcement learning. Rudin et al. use a position-based formulation that avoids some of the pitfalls of tracking a reference velocity from a joystick like poor energy efficiency and they show complex behaviors like jumping over gaps though their approach is prone to training instability and the robot learns to walk in one direction only. Zhuang et al. (2023) show another example using deep RL and a two-stage training process to learn 5 skills including climbing and leaping, deploying on a real robot and showing high success rates on gaps more than 1.5x the length of the robot, though the two-stage process adds complexity to training and the training environment had to be manually constructed, limiting new skill acquisition. This does not meet the objective of simplifying skill acquisition and generalize-ability to new obstacles. Hoeller et al. (2024) use a hierarchical fully-learned approach with a perception module, navigation module, and skill module, though this comes at the cost of complexity with 8 networks to train and difficulty in adding new skills. Again, this does not meet the objective of easily learning new skills. Cheng et al. (2024) have a single policy for many skills that they achieve using a unified reward function, careful choice of scandots as privileged information, and then distillation. However, they also have a limited set of 4 obstacles they train on and demonstrate skills with. Huang et al. (2025) is a pre-print that explores using a mixture-of-experts policy to learn a diverse set of skills with a two-stage training process where the first stage uses privileged data, though they rely on proprioceptive sensor data and do not integrate sensor data like lidar or camera images. Rudin et al. (2025) is a pre-print that explores using a mixture of experts via distillation and data aggregation (DAgger), followed by RL fine tuning that shows great generalizeability. This was released a week or two before the final report, without code, but it is what we wanted to do and better. That said, they only show performance on one type of robot, rather than a multibody solution.

From the literature review, the key themes are that successful approaches in locomotion on challenging terrain, once trained, may struggle to add new skills and generalize. Also, often a training environment with a limited set of obstacles is used and this may hurt generalization in real world tasks like navigating a garden that wasn't in training data. The existing methods therefore do not meet the objectives. Our method is novel because it targets generalization and using visual input with a mixture of experts.

3 Method

3.1 Training Environment

We use the legged gym environment from Cheng et al. (2024) including their observation structure, reward structure, and parallelized steps. Figure 1 shows the training environment which encompassed 4 types of parkour obstacles. As success rate increased, the terrain is made more difficult by the environment automatically. At 8k steps the environment enters a domain randomization phase where it randomizes physical parameters like friction among others, making the training more difficult. Note that the obstacles involve many skills like jumping, climbing up and down, and stepping-over. These may cause conflicts when trying to learn. The observation mimics that of Cheng et al. (2024) and includes proprioceptive data like joint position, velocities, and torques, as well as command velocities. It also offers privileged data that Cheng et al. used in a two phase training approach. This includes positions of waypoints for each obstacle, heading to the waypoint, and scandots instead of depth images. We do just the first phase of this training and use priviledged data, though we do train estimator networks as well to transition to phase 2. The action space is 12 dimensional and is joint positions for each joint on the quadruped. The reward is the same as that in Cheng et al. (2024). We simulate the Unitree A1 robot. We do 24 timesteps in our collection phase and at any point in these timesteps a given environment may terminate and reset. So while our batch will always have 24 timesteps, care has to be taken to treat termination appropriately.

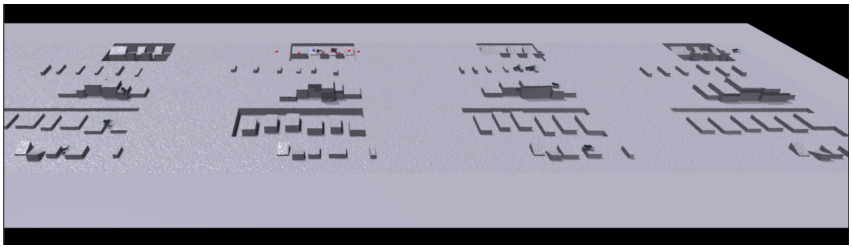

Figure 1: Training environment with variety of obstacles.

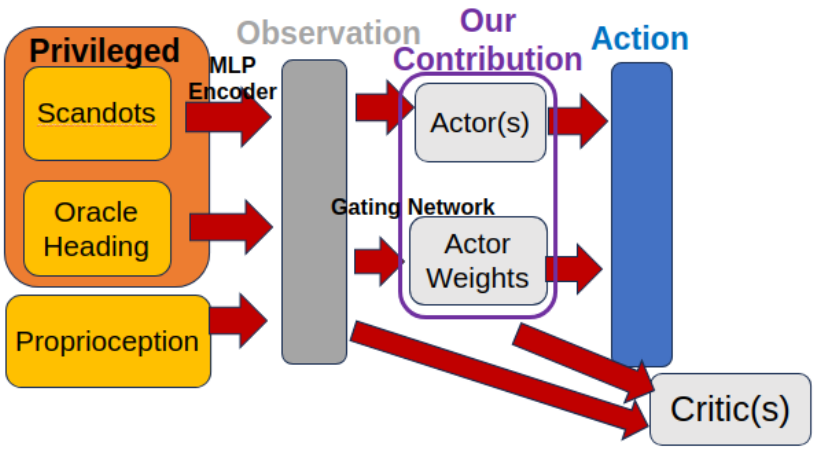

Figure 2: Observation, Network Architecture, and Actions.

3.2 Network Architecture

We have similar architecture to Cheng et al., up until the actor which we replace with the mixture of experts (MoE). We briefly review this architecture here, and discuss the MoE in a seperate subsection. Figure 2 shows an overview of networks used. Not pictured are some of the behind the scenes networks that aid in the transition from Phase 1 (using privileged information) to Phase 2 (estimating privileged information). We don't focus on reviewing this architecture here because in this paper we only do Phase 1 training and testing, due to time constraints. However, an example of other modules (that we do train, simply we don't use in phase 2) includes an estimator that takes in proprioceptive data and estimates implicit privileged data using supervised learning that represents things like waypoints and heading to waypoints.

3.3 PPO Algorithm

We implement a version of proximal policy optimization (PPO). Namely, we:

1. Sample a batch of data from π_θ
2. Train $\hat{V}_\phi^{\pi_\theta}$ using bootstrapping and supervised learning where $y_{i,t} = r(s_{i,t}, a_{i,t}) + \hat{V}_\phi^{\pi}(s_{i,t+1})$
3. Calculate advantages by evaluating $\hat{A}^\pi(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}_\phi^{\pi}(s_t) + \gamma^n \hat{V}_\phi^{\pi}(s_{t+n})$, where n is the number of timesteps in our batch, taking care to handle terminations by zeroing out the value at the next time step if terminated given the rolling buffer.
4. Update policy using importance weights by evaluating)

$$\tilde{J}(\theta') \approx \sum_{t,i} min\left(\frac{\pi_{\theta'}(a_{i,t}|s_{i,t})}{\pi_\theta(a_{i,t}|s_{i,t})}\hat{A}^\pi(s_t, a_t), clip(\frac{\pi_{\theta'}(a_{i,t}|s_{i,t})}{\pi_\theta(a_{i,t}|s_{i,t})}, 1-\epsilon, 1+\epsilon)\hat{A}^\pi(s_t, a_t)\right)$$

In contrast to Cheng et al, we use a separate optimizer for the critic whereas they combine it with the actor in one optimizer. They also adjust the learning rate by comparing KL divergence to a desired KL divergence which we skip. They also clamp the value function updates which we do not. We produce a modified version of their codebase to use as a second baseline (in addition to the first), that does not clip the value function and does not use KL divergence to adjust the learning rate, so as to make comparisons easier.

3.4 Mixture of Experts

We use a multi-layer perceptron as the gating network to produce weights for our cohort of experts. It has input dimensions equal to those of the observation, output dimensions equal to those of the number of experts, and hidden layer dimensions of [256, 128]. Shazeer et al. (2017) make suggestions including sparsity and adding a term to the objective to include diversity, which we implement. Specifically, equation (1) shows how we get intermediate weights where $q(x)$ is the output of the MLP gating network, W_{noise} is a trainable noise parameter. The standard normal encourages load balancing as Shazeer et al. show. Equation (2) uses the $KeepTopK$ where for each observation, only the k highest outputs from $H(x)_i$ are used, the others are set to $-\infty$ and softmax is used to get a valid categorical probability distribution of the k experts. The nan values are set to zero for unused experts. This is to encourage sparsity which can help when scaling the number of experts, and in experiments and results we show the impact of this hyperparameter. The output of this are weights for the expert actors $G(x)$ which may be zero or non-zero.

$$H(x)_i = q(x)_i + StandardNormal() \cdot Softplus(W_{noise}) \tag{1}$$

$$G(x) = Softmax(KeepTopK(H(x), k)) \tag{2}$$

The expert actors are each MLPs, and take as input the observation and output the means for a gaussian probability distribution, where each actor has a trainable parameter for the log standard deviations of the distribution. The actor architecture is the same as that used by Cheng et al, namely hidden dimensions of [512, 256, 128]. We use a mixture of gaussians model to combine these as equation (3) shows where N is the number of experts and π_n is the processed probability from the gating network for an individual expert. We use pytorch to do this using the MixtureSameFamily class.

$$\mathcal{N}(\mu_k, \sigma_k) = \sum_{n=1}^{N} \pi_n \mathcal{N}(x|\mu_n, \sigma_n), 0 \leq \pi_n \leq 1, \sum_{n=1}^{N} \pi_k = 1 \tag{3}$$

We also introduce a term into the optimization to encourage diversity of experts, given MoEs can favor a limited number of experts, which we take from Shazeer et al. (2017). We calculate the importance of an expert as the sum of its weights across the batch as equation (4) shows and then calculate the coefficient of variation of these importances and weight this with a manually tuned coefficient $w_{importance}$ as equation (5) shows.

$$Importance(X) = \sum_{x \in X} G(x) \tag{4}$$

$$L_{importance}(X) = w_{importance} \cdot CV(Importance(X))^2 \tag{5}$$

The idea behind the mixture of experts is that each expert may learn responsibility for a different component of the environment like stepping down or jumping, and that upon seeing new obstacles not in the training data the gating network may be able to combine experts to mimic the new obstacle and create something that works better than a single actor may be able to output.

4 Experimental Setup

For the experiment, we train 4 algorithms and compare metrics during training. The original algorithm we used as a baseline was trained on an Nvidia 3090 GPU and uses many parallel environments.

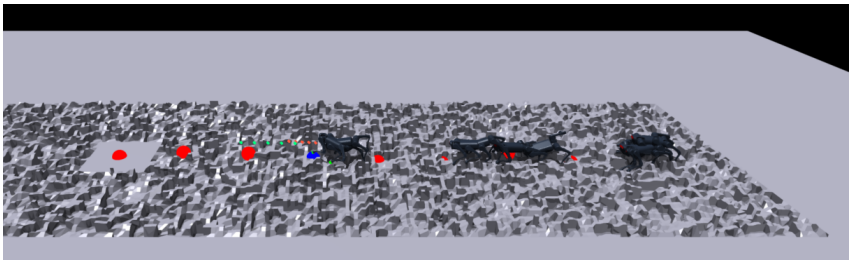

Figure 3: Obstacle we created to test generalization that is dissimilar to training data seen by the networks. Pictured is the modified baseline, failing to reach the goal in the middle.

The batch size is the number of parallel environments multiplied by the number of time steps per collection (fixed at 24), which is then split into mini batches. We made an effort to keep batch size consistent across the last 3 algorithms, but we also compare with the original algorithm that used the large batch size and many environments. These algorithms are:

1. Baseline algorithm from Cheng et al. (2024) with no code changes. This uses 6144 environments, which means a large batch size. They use PPO with some enhancements like using KL divergence and desired divergence to adjust learning rate. They use a mini-batch size of 4.
2. Baseline algorithm from Cheng et al. (2024) with code changes to simplify their PPO algorithm to be comparable to ours. Namely we remove the adjustment of the learning rate using KL divergence and we do not clamp the value function updates. We also lower the number of environments to 2048, which is what can fit on our local 2060 GPU. This has an impact on the batch size given our training setup. We use 4 mini-batch size.
3. The MoE approach using 2048 environments, 1 mini-batch, 5 experts, and no sparsity (all expert weights are used).
4. The MoE appraoch using 2048 environments, 4 mini-batches, 5 experts, and the top 2 experts for weights.

We choose (1) as the baseline as it is a well reviewed work, uses a depth camera like we do, and shows good performance both in simulation and in real-life by jumping more than 2x the robot's length.

We test (4) and compare it to (3) because using only the top k experts, with a limited set of experts, means that one expert may get favored at the cost of others. This has the effect of "snowballing", meaning that the favored expert improves even more quickly as more of a batch is directed to it and it can improve further from that. Hence we experiment with (3) and (4) to compare the impact of this hyperparameter.

We also trained one initial mixture of experts without the diversity objective included, and we show the results of this in comparison to the MoE approaches with the diversity objective to examine its importance.

We also test all four algorithms on a new obstacle that we created that is unseen and dissimilar to the obstacles that the algorithms were trained with. This obstacle was intended to resemble a rocky field, and figure 3 shows this obstacle. This experiment is to test how well each approach generalizes to new obstacles and data that were not seen in the training phase. We spawn 10 robots for each algorithm and start a timer for 75 seconds. We record the number of successful robots expressed as a percent of the total robots.

The metrics we report for training include mean terrain level across recently completed runs. The environment up-levels the terrain by making obstacles higher or gaps longer, as more environments succeed. Therefore the level of terrain is a useful metric for how "good" a policy is. We also look at mean reward across a batch, useful to determine how well we meet the environments objectives. We also look at episode length briefly, useful in the early stage of training when robots fall over and reset. We use this as a gauge to tell if algorithms are learning successfully.

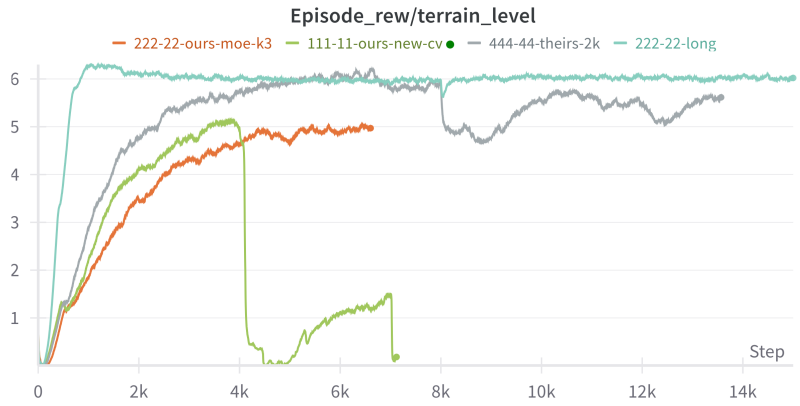

Figure 4: Terrain level of the four algorithms with unmodified baseline in the light blue, modified baseline in grey, the MoE approach using all experts in orange, and the MoE approach using sparsity and the top 2 experts in green.

Table 1: Performance comparison of algorithms. The train numbers are given for 6k training steps. The new obstacle success rate uses 14.5k steps for the baseline, 13k steps for the modified baseline, 6k steps for MoE without Sparsity, and 7k steps for the MoE with Sparsity.

Method	Train Mean Terrain Level	Train Mean Reward	New Obstacle Success Rate
Baseline	5.97	15.61	0%
Modified Baseline	6.05	9.91	0%
MoE without Sparsity	4.9	10.91	30%
MoE with Sparsity	1.1	4.40	0%

5 Results

5.1 Quantitative Evaluation

See table 1, 2, and figures 4, 5, 6.

5.2 Qualitative Analysis

We first analyze the performance of the 4 algorithms with respect to training and generalization. The MoE approaches train more slowly, yielding less terrain level for a given number of steps, as figure 4 and table 1 show. This is not unexpected, because training many actors, especially when splitting data to them using sparsity or small mini batches, can take longer to improve. There is a significant performance gap between the MoE approaches that use sparsity and those that don't. In the context of this problem, the number of experts is quite limited. By forcing only 2 experts for each observation, the amount of training data being passed to other experts is quite limited and improvement is slow. This also makes it easier for policies to rely too heavily on one set of experts, and the figures show that this can result in crashes and slow recovery. In fact, table 2 analyzes the impact of over-reliance on one expert caused by no diversity objective which can cause highly unstable learning and crashes.

Table 2: Performance comparison of MoE approaches analyzing the importance of diversity objectives. Metrics pulled at convergence for the policy without diversity (7.5k steps), at 6k for the policy with diversity (when training crashed due to spot instance terminating).

Method	Min Mean Expert Weight	Max Mean Expert Weight	Mean Terrain Level
MoE w/ sparsity, w/ diversity	0.12	0.25	0.4
MoE w/ sparsity, w/o diversity	9e-28	1.0	0.0

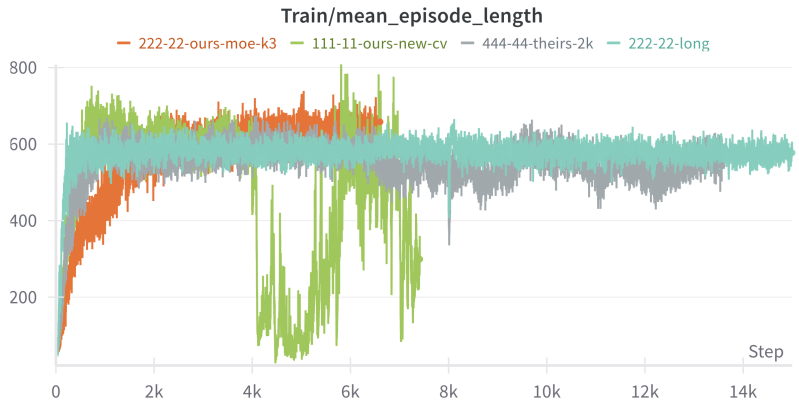

Figure 5: Mean Episode length for the four algorithms, with unmodified baseline in the light blue, modified baseline in grey, the MoE approach using all experts in orange, and the MoE approach using sparsity and the top 2 experts in green. Shows that MoE approaches take longer to train.

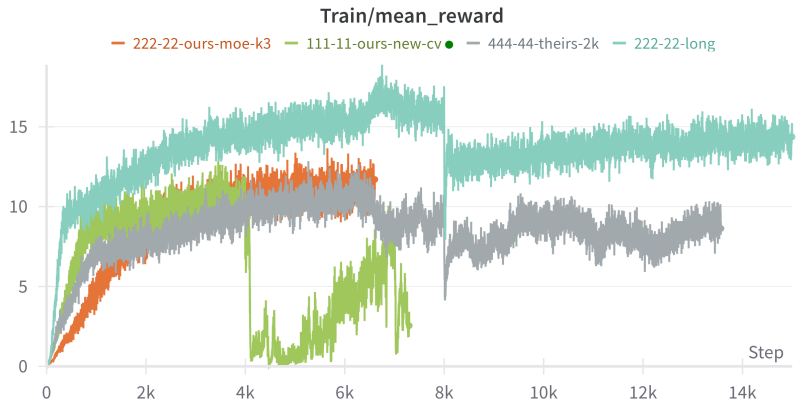

Figure 6: Mean Reward for the four algorithms, with unmodified baseline in the light blue, modified baseline in grey, the MoE approach using all experts in orange, and the MoE approach using sparsity and the top 2 experts in green. Shows that the unmodified baseline outperforms.

Looking at reward, figure 6 shows that the unmodified baseline outperforms, possibly due to its tweaking of learning rates and clamping. The picture is more mixed for the modified baseline compared to the MoE without sparsity.

Looking at generalization, the MoE without sparsity is the clear winner as it was the only policy that found some success, albeit low success with 30%. It's also interesting to note it was the policy trained with the least number of steps (6k).

One final interesting note is on computational cost and batch size. Shazeer et al. (2017) note that higher batch sizes are helpful for MoE given it amortizes the cost of parameter transfer and updates. However, this comes at the cost of memory. On memory constrained systems like our local GPU (Nvidia 2060), we couldn't do a full batch at once without hitting memory limits. Hence we trained some of the algorithms that used large batch sizes in AWS. This may be a consideration on more limited systems.

6 Discussion

The results from this study are interesting to further generalization of RL policies to new obstacles unseen in the training data. However, this study was conducted only on phase 1 training, using privileged data. It should be repeated after fine tuning policies using phase 2 training, depth images and estimators. Additionally, the MoE policy only achieved 30% success on new obstacles. While this is better than other policies in absolute terms it is still quite poor. Also, deployment on a real robot is missing from this study.

However, this study has the potential for impact, as quadruped parkour is a very active field of research shown by the number of papers in the past 2 years in the related works section (5!).

7 Conclusion

In this study we examine the impact of using a mixture of experts in the context of quadruped parkour, as a tool to further quadrupedal locomotion on challenging terrain. We find evidence that MoEs can help generalize RL policies to previously unseen obstacles, though more work is needed without privileged data and using real robots. In general, RL is a powerful tool that we explored but it's limitations need to be kept in mind. What works in simulation may not work in real life on real robots due to physical parameter assumptions and dynamics assumptions made in simulators. Future work in this direction can integrate real world training data on obstacles, to bridge the sim-to-real gap and perhaps further improve generalization.

8 Team Contributions

- **Michael Ziegltrum:** did all the work in this project.

Changes from Proposal The allocation of work didn't change. We are a group of 1.

References

Marko Bjelonic, Prajish K Sankar, C Dario Bellicoso, Heike Vallery, and Marco Hutter. 2020. Rolling in the deep–hybrid locomotion for wheeled-legged robots using online trajectory optimization. *IEEE Robotics and Automation Letters* 5, 2 (2020), 3626–3633.

Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. 2024. Extreme parkour with legged robots. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11443–11450.

David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. 2024. Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics* 9, 88 (2024), eadi7566.

Runhan Huang, Shaoting Zhu, Yilun Du, and Hang Zhao. 2025. MoE-Loco: Mixture of Experts for Multitask Locomotion. *arXiv preprint arXiv:2503.08564* (2025).

Sotaro Katayama, Masaki Murooka, and Yuichi Tazaki. 2023. Model predictive control of legged and humanoid robots: models and algorithms. *Advanced Robotics* 37, 5 (2023), 298–315.

Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. 2019. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. *arXiv preprint arXiv:1909.06586* (2019).

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. 2020. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784* (2020).

Nikita Rudin, Junzhe He, Joshua Aurand, and Marco Hutter. 2025. Parkour in the Wild: Learning a General and Extensible Agile Locomotion Policy Using Multi-expert Distillation and RL Fine-tuning. *arXiv preprint arXiv:2505.11164* (2025).

Nikita Rudin, David Hoeller, Marko Bjelonic, and Marco Hutter. 2022. Advanced Skills by Learning Locomotion and Local Navigation End-to-End. arXiv:2209.12827 [cs.RO] https://arxiv.org/abs/2209.12827

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

Ziwen Zhuang, Zipeng Fu, Jianren Wang, Christopher Atkeson, Soeren Schwertfeger, Chelsea Finn, and Hang Zhao. 2023. Robot parkour learning. *arXiv preprint arXiv:2309.05665* (2023).