Multiple-output genetic programming tree for Gym problems

Makar Shevchenko
B20-DS-01
Innopolis University
Innopolis, Russia
m.shevchenko@innopolis.university

Aliaksei Korshuk
B20-DS-01
Innopolis University
Innopolis, Russia
a.korshuk@innopolis.university

Viacheslav Sinii
B20-DS-01
Innopolis University
Innopolis, Russia
v.sinii@innopolis.university

***Abstract*—This work evaluates multiple-output genetic programming (GP) tree as a tool for building interpretable models for reinforcement learning (RL) environments. We benchmark considering approach with a GP tree-per-output model and PPO neural network model on Gym RL environments. We found out a limitation of a method on pixel games. The observations and analysis are present.**

I. Introduction

Reinforcement learning is a problem, a class of solution methods and research field about mapping state to the action in order to maximise reward without direct instructions [1]. RL as a problem has several tricky properties that makes learning difficult. First, current decisions affect future state and reward. So, it might happen that although one of the actions at the moment will provide the greatest reward, in the long run, the alternative behavior will be more profitable. So, agents should balance between short-term and long-term planning. Second, at the beginning agents' knowledge is limited, they do not know what reward they can get in some cases. However, it might beneficial to discover search space in order to find better conditions. So, this property makes agents balance between exploration and explointation. Listed difficulties make RL an interesting area of research.

Neural networks are heavely used for RL problems for their outstanding performance. However often the solution of a neural network is a black-box. It is a black-box in the sense that it is uninterpretable due to the high complexity. Complexity arises due to use of thousands or even millions of parameters that form complex non-linear relationships between inputs and outputs. Non-interpretability raises concerns for the use of such solutions in medicine and other areas requiring security guarantees. Additionally, this property is a barrier to extract insights about effective behavior in the RL environment [2] [3].

In contrast, GP is a different approach for RL tasks that builds interpretable solutions [4]. Solution of GP is a computer program that is usually sufficiently simple to be computed manually. Therefore, GP is a more attractive method in terms of obtaining insights about effective behavior and guarantees from the solution.

GP works by a principle of natural selection among computer programs [4]. Programs are built from a set of predefined operators, such as summation, multiplication, and branching statements. Programs evolve at runtime, changing under the influence of mutation and genetic crossover. As a result of selection, the population becomes more and more adapted to the problem. Finally, the best individual is chosen as the solution.

In this work we show comparison between two types of multi-objective GP methods and one neural network based method. Results demonstrate that GP methods outperform neural networks in training speed and even outperform them on some environments. Next, we show how the solutions of GP methods can be explained in contrast to neural networks. Finally, we contributed our implementation of multiple-output GP tree to open-source library for gentetic algorithms DEAP.

II. Related work

Zhang et al [5] proposed an architecture of multiple-output GP tree. The idea is to use a special operator that add computed value to one of the outputs. We implemented proposed model in our work.

Videau et al [6] compared multi-objective GP approaches with neural network ones. Results show that GP approach may outperform neural networks in some cases, especially with delayed reward. However, it do not consider solution from [5].

Zhang et at [2] and Hein et al [3] showed a training technique for a GP tree-per-output model based on approximation behavior projectory. Project trajectory is generated by a pretrained neural network.

III. Methodology

A. Models

In most simple games the mapping from a state to an action can be expressed as closed-form function. It is a natural application of genetic programming and we leverage this technique to find the exact formula.

1) Single Action Space: Genetic Programming is naturally applicable here. A mathematical formula can be expressed as a tree where root is the result of calculations, internal nodes are operations and terminal nodes are either the input variables (state of the game in our case) or functions without variables such as constants and random number generators.

Decision Making. For binary actions (do or do not do) we make a decision by checking whether the output is greater (do) or less (don't do) than zero. For continuous actions, such as the speed of a car, we return the output as it is.

Fitness Function. We obtain the fitness by taking the reward after running our agents in a Gym.

2) Mutliple Action Space: Evolution of the usual tree doesn't scale to games with multiple outputs because it returns only single number. For that reason, we implemented modified individuals which return vector of outputs. For discrete games we apply argmax function and return the result as an action. In games with continuous actions we return the result unaltered.

Modi The initial methodology is described in [5], and we implemented this idea with a slight modification. The authors suggest to add a special node which passes the result of their calculations to the parent (as usual), but also adds this result to the output vector. Each such node has an assigned number which specifies the index to which it will add the result.

Instead, we decided to separate these two functions. We add a special node called 'modiindex' which passes its input to the parent without changes and adds this input to the output vector. This approach allowed us to simplify the implementation.

Multi-Tree

The idea is to create a bag of trees where each one is responsible for specific output index. Thus, for output vector with size N we have N populations. To obtain an action, we take i-th individual from each population, feed them the state of the game and collect outputs.

IV. GitHub Link

We published the code of evaluation tests and implementation of tree-per-output model in our GitHub repository by the following link: https://github.com/AlekseyKorshuk/dgp. Implementation of multiple output RL tree is published in our fork of DEAP repository by the following link: https://github.com/SyrexMinus/deap_MultiOutputTree.

We contributed implemented model to open-source library for gentetic algorithms DEAP via the pull request by the following link: https://github.com/DEAP/deap/pull/678.

V. Experiments and Evaluation

A. Experiments configuration

We compare the performance of the proposed approaches to the PPO RL agent.

We evaluated models based on rewards obtained on the following Gym environments:

- with discrete output: CartPole-v1, Acrobot-v1, MountainCar-v0, LunarLander-v2, FlappyBird-v0, Pixelcopter-PLE-v0, and Pong-PLE-v0;
- with continuous output: MountainCarContinuous-v0, Pendulum-v1, and BipedalWalker-v3.

We evaluated results depending on the training time that was measured both it in seconds and number of environment runs.

Training configurations for models are:

- Population size 30, 30 generation, 1 environment run per individual for GP agents;
- Number of environment timestamps for whole training is 25000 for PPO agent.

This configuration was empirically derived to have approximately the same training time in environment runs for all the models.

Experiments were conducted on Apple M1 Pro with 10-core CPU workstation.

We used Python 3.8 programming language for evalution tests, DEAP framework to implement GP methods, and an implementation of PPO [7] from Stable-Baselines3 library [8] based on PyTorch framework [9].

B. Results of Experiments

From the results shown on figures 1, 2, and 3 we noticed that GP agents fit much faster than PPO agent. Moreover final reward of the GP agents is usually bigger compared to PPO.

VI. Analysis and Observations

A. On Performance

GP get greater awar compared to PPO model due to simplicity of considered environments. Most of them require proper reaction on in-game physics. GP methods deal well with such a task due to its struture.

B. On Interpretability

Usage of genetic programming allows to make the results interpretable because in the end of training we have a closed-form function with all operations clearly visible and explainable. That may be useful in applications where understanding of agent's policy is crucial.

C. Limitations

1) Pixel States: When we feed a pixel snapshot of a game as an input for a model, GP faces a problem similar to the difficulties of applying classical machine learning to Computer Vision problems. The complex nature of pictures makes it hard for a simple model to extract good features and build a closed-from function with their help. Application of GP in these types of games requires further research.

VII. Conclusion

In this work we implemented and evaluated two GP models: multiple-output GP tree and GP tree-per-output. We evaluated and compared their performance with PPO model and found out that GP methods outperform PPO in training time and give better final results. Moreover, solutions found by GP suggest optimal behavior for RL environments. However, we found that GP methods cannot deal with pixel games since it requires computer vision abilities.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Hengzhe Zhang, Aimin Zhou, and Xin Lin. Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis. *Complex & Intelligent Systems*, 6(3):741–753, 2020.

[3] Daniel Hein, Steffen Udluft, and Thomas A. Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76:158–169, 2018.

[4] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

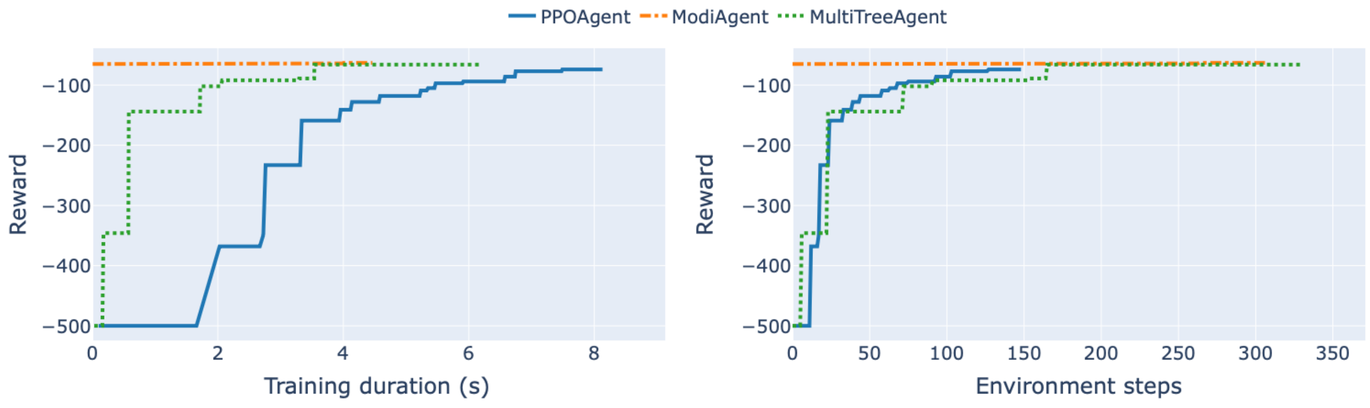
[5] Yun Zhang and Mengjie Zhang. A multiple-output program tree structure in genetic programming. pages 6–10, 01 2005.

[6] Mathurin Videau, Alessandro Leite, Olivier Teytaud, and Marc Schoenauer. Multi-objective genetic programming for explainable reinforcement learning. In Eric Medvet, Gisele Pappa, and Bing Xue, editors, *Genetic Programming*, pages 278–293, Cham, 2022. Springer International Publishing.

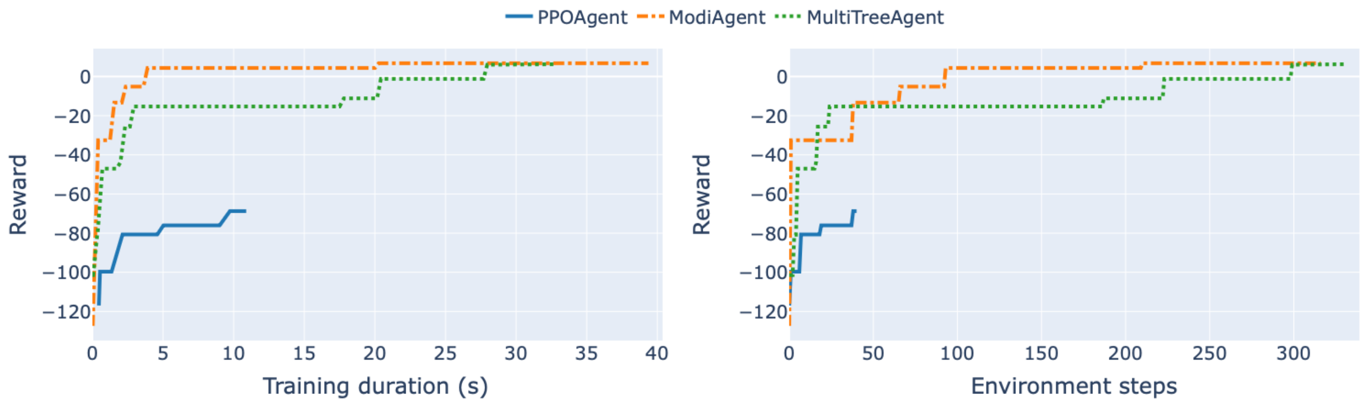
[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[8] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

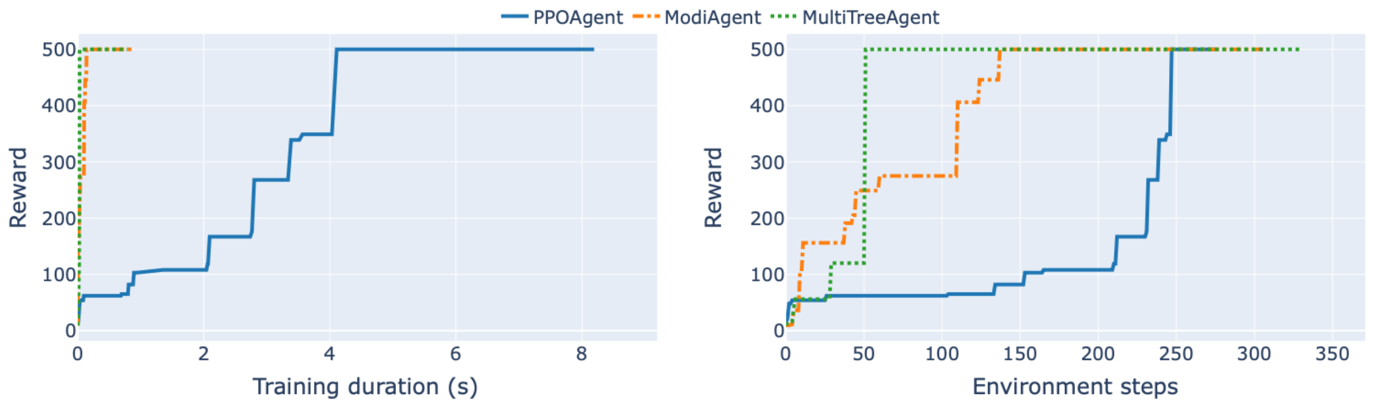
[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.



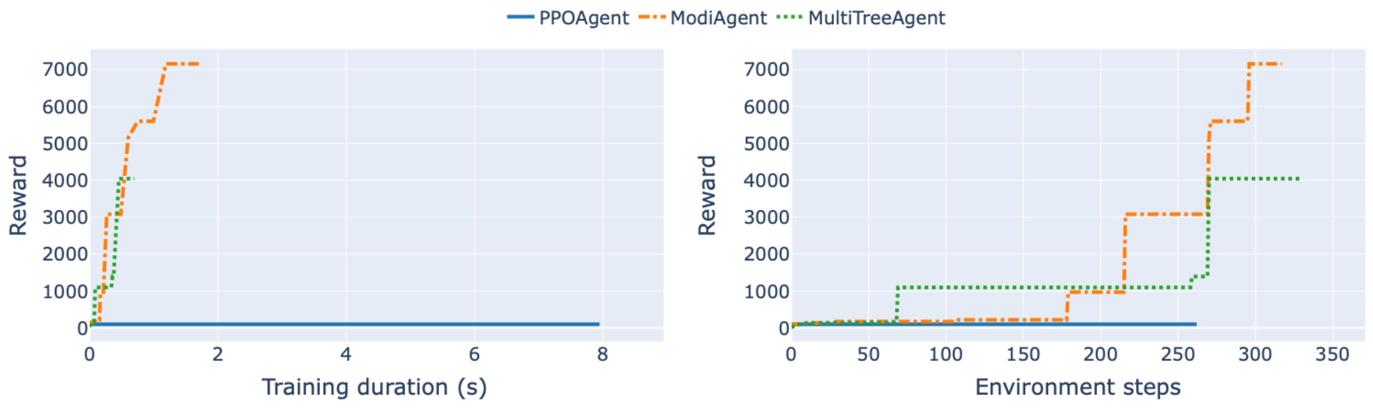
(a) Acrobot-v1 environment



(b) BipedalWalker-v3 environment

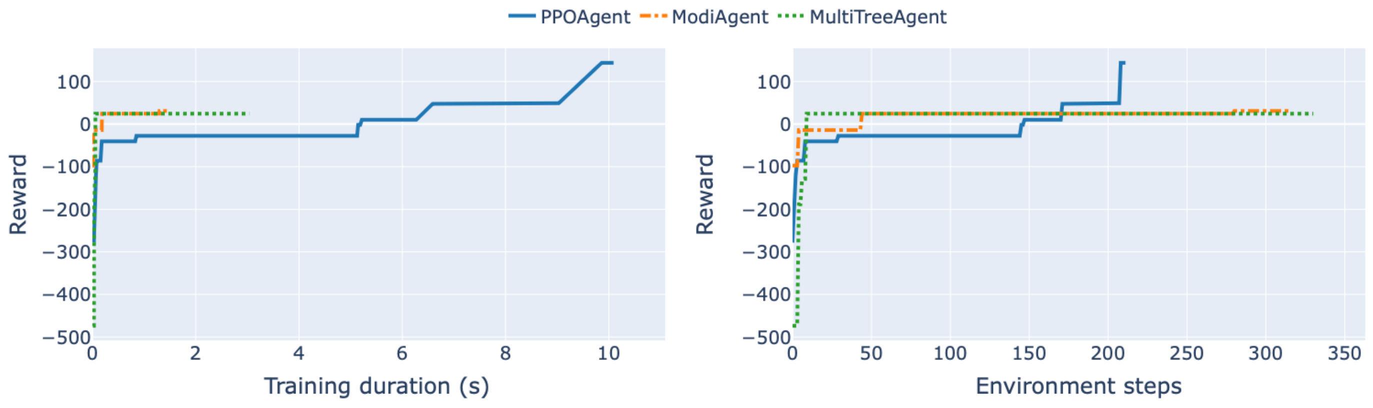


(c) CartPole-v1 environment

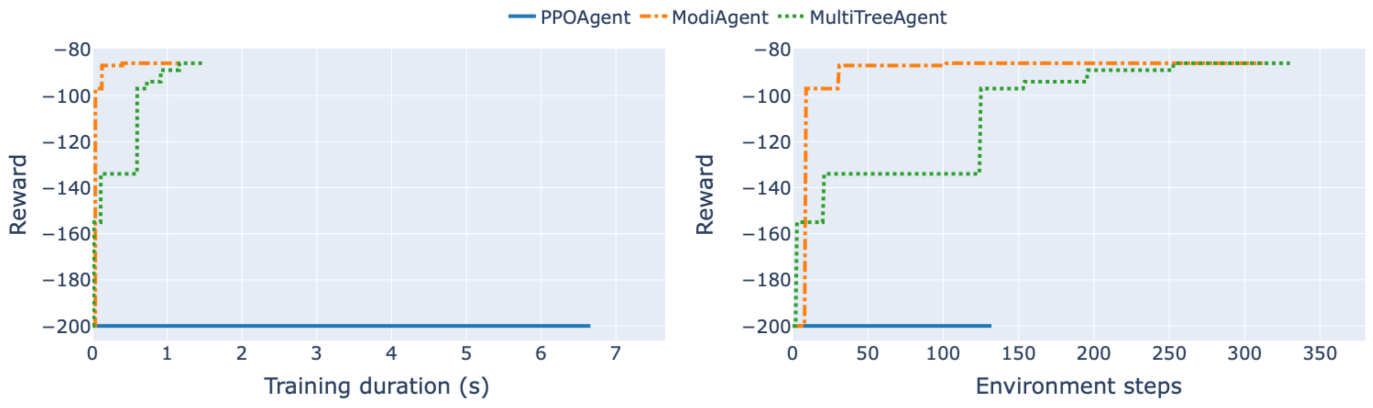


(d) FlappyBird-v0 environment

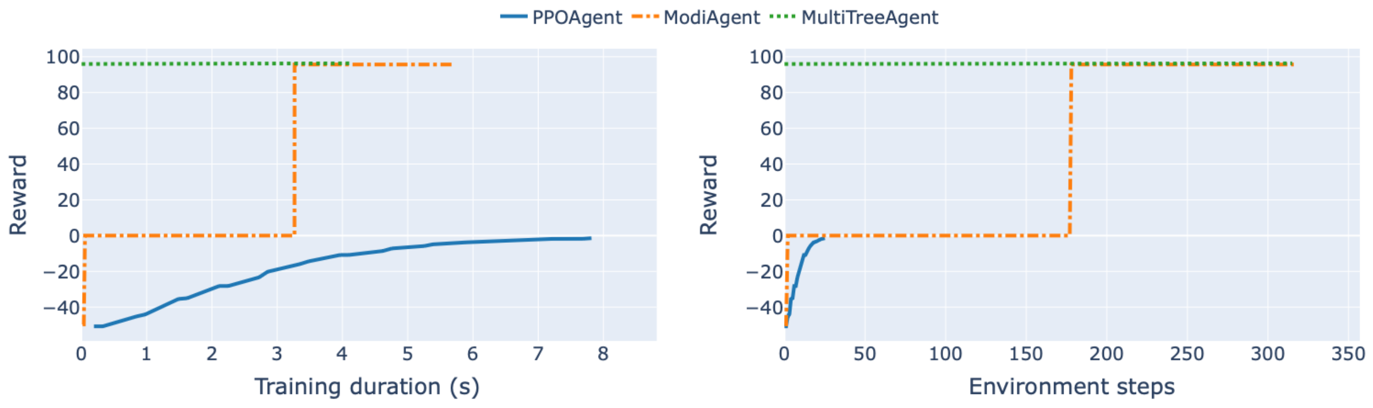
Fig. 1: (Part 1) Evaluation of maximum reached rewards by agents vs time of training for different environments. Evaluated models are PPO, multiple-output GP tree (ModiAgent), and GP tree-per-output (MultiTreeAgent). Time of training represented in seconds on left figures, and in number of environment executions on right figures.



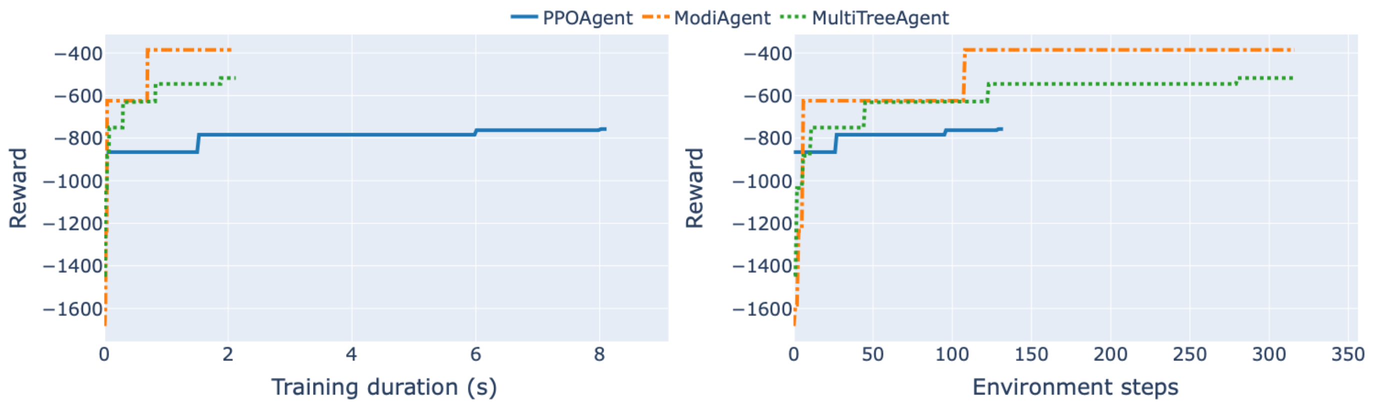
(a) LunarLander-v2 environment



(b) MountainCar-v0 environment

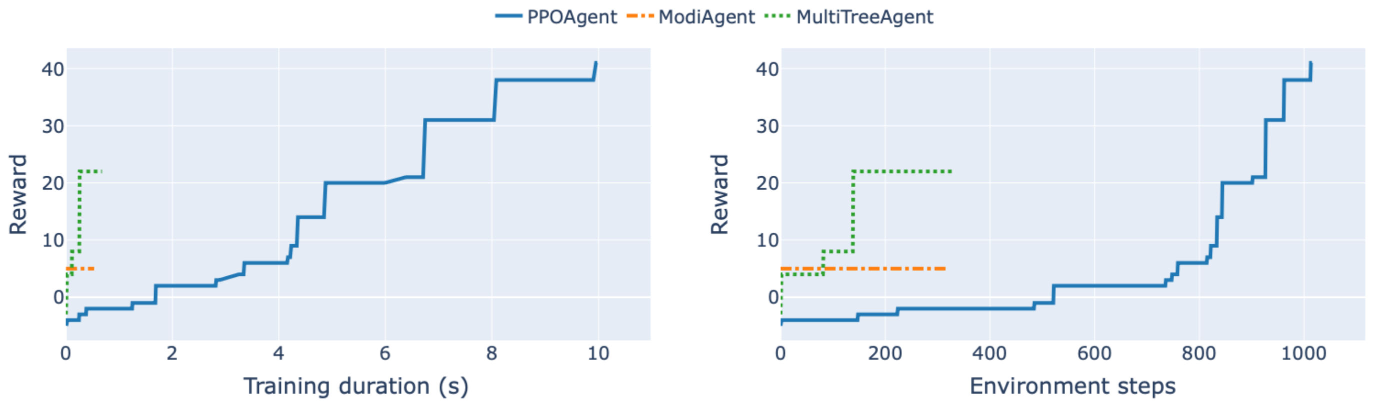


(c) MountainCarContinuous-v0 environment

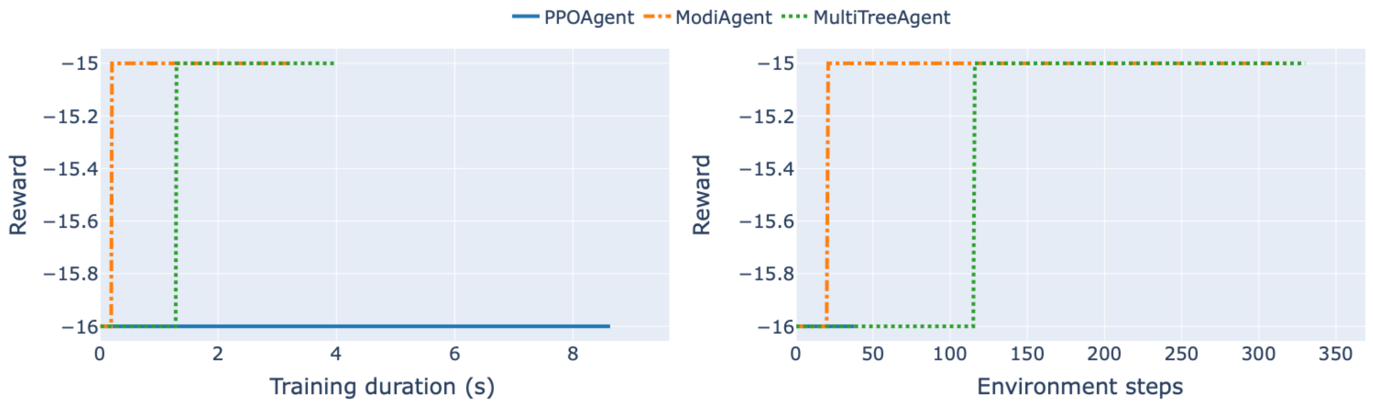


(d) Pendulum-v1 environment

Fig. 2: (Part 2) Evaluation of maximum reached rewards by agents vs time of training for different environments. Evaluated models are PPO, multiple-output GP tree (ModiAgent), and GP tree-per-output (MultiTreeAgent). Time of training represented in seconds on left figures, and in number of environment executions on right figures.



(a) Pixelcopter-PLE-v0 environment



(b) Pong-PLE-v0 environment

Fig. 3: (Part 3) Evaluation of maximum reached rewards by agents vs time of training for different environments. Evaluated models are PPO, multiple-output GP tree (ModiAgent), and GP tree-per-output (MultiTreeAgent). Time of training represented in seconds on left figures, and in number of environment executions on right figures.