Unity ML Agents vs. Godot RL Agents

Comparing the performance of two RL frameworks

Florentin Luca Rieger, BSc[1], Edward Beeching, PhD [2], Aleksei Petrenko, PhD [3], Markus Petz, MSc[1], FH-Prof. Dipl.-Ing. Alexander Nimmervoll[1]

[1] University of Applied Sciences Technikum Wien, Vienna, Austria

[2] Hugging Face, Lyon, Auvergne-Rhône-Alpes, France

[3] Apple, Mountain View, California, United States

florentin.rieger@gmail.com

Abstract

Over the last years, a lot of different reinforcement learning tools have emerged. From libraries, which provide high quality, well tested RL algorithms, to frameworks, which, additionally, allow to create simulations for RL problems, train agents inside those simulations and visualize the learning process. This work compares two of those RL frameworks: Unity ML Agents and Godot RL Agents in terms of learning- and computational runtime performance. The goal is to provide ML researchers and game developers some insights, which tool might fit their needs best. Unity ML Agents provides its own custom RL algorithms, while Godot RL Agents offers 4 different RL backends (CleanRL, Ray RLlib, Stable Baselines3, Sample Factory) with their implementations of RL algorithms. To evaluate the performance of the two frameworks and 4 RL backends, two different trials were conducted, in which the agents were trained on a small problem. One trial was run under the frameworks' default hyperparameter configuration, and the other trial was executed under a shared configuration, with a set of identical hyperparameters. The trials under the default hyperparameter configuration revealed that Godot RL Agents[SF] and Unity ML Agents achieved the most return, but Godot RL Agents[SF] learned up to 46 times faster than Unity ML Agents in the first minutes of training. The trials under the shared hyperparameter configuration showed, that Godot RL Agents[SF] trained much worse than before, indicating its sensitivity to hyperparameter changes, while Unity ML Agents performed even better than previously. The other backends (CleanRL, SB3) performed reasonably well under both configurations, except for RLlib which was not able to learn at all (default configuration) or achieved relatively small rewards (shared configuration). Furthermore, the results indicate the importance of custom fitted hyperparameter configurations as implementation details seem to differ a lot.

Introduction

Artificial intelligence (AI), especially in the field of machine learning (ML), has made huge advances in the last 10 years. From large language models (LLMs) [1], self-driving cars [2], prediction of protein structures in the field of medicine and biology [3], to high performing agents in computer games [4], artificial intelligence has had major impacts in almost every aspect of life. The introduction of deep artificial neural network (ANN) and their combination with other machine learning algorithms has made endeavours possible which have been thought of as impossible previously [5].

One of those advancements is deep reinforcement learning (DRL) [6], which is a combination of reinforcement learning (RL) algorithms and deep artificial neural networks. These DRL algorithms are self-learning, sequential decision-making systems and are most known for performing tasks in the

fields of robotics, self-driving cars, healthcare, finance, advertisement, large language models [7] and games [8]. Over the last years, following those advancements, a lot of different reinforcement learning tools have emerged. From libraries, which provide high quality, well tested RL algorithms [9] [10] [11] [12], to frameworks, which, additionally, allow to create simulations for RL problems, train agents inside those simulations and visualize the learning process [13] [14]. The goal of this work is to provide ML researchers and game developers insights about which tool might fit their needs best, by comparing two RL frameworks: Unity ML Agents [13] and Godot RL Agents [14] in terms of their learning- and computational runtime performance.

Unity ML Agents is an open source RL framework by Unity for their proprietary game engine. The framework provides its own custom implementations of popular on- and off-policy reinforcement learning algorithms like Proximal Policy Optimization (PPO) [15] and Soft Actor Critic (SAC) [16]. Godot RL Agents is an open source RL framework by Edward Beeching [14] for the open source game engine Godot. It offers the choice between four different RL backends (CleanRL [9], Ray RLlib (RLlib) [10], Stable Baselines3 (SB3) [11] and Sample Factory (SF) [12]), each with their own implementations of various RL algorithms. This results effectively in the comparison of 5 different frameworks: Unity ML Agents, Godot RL Agents[CleanRL], Godot RL Agents[RLlib], Godot RL Agents[SB3] and Godot RL Agents[SF].

The research question is formulated as: “How do the reinforcement learning frameworks Unity ML Agents and Godot RL Agents differ in terms of learning- and computational runtime performance?”.

Methods

In order to compare the two different frameworks, a simulation was built in Unity and Godot and the agents were trained inside those simulations with the PPO algorithm [15] under two different hyperparameter configurations. The simulation is a small toy problem, in which the avatar, controlled by the agent, has to jump from platform to platform and collect as many coins as possible without falling down. Every time the agent collects a coin, a new platform and coin gets spawned at random position. If the agent falls of a platform the episode ends and restarts.

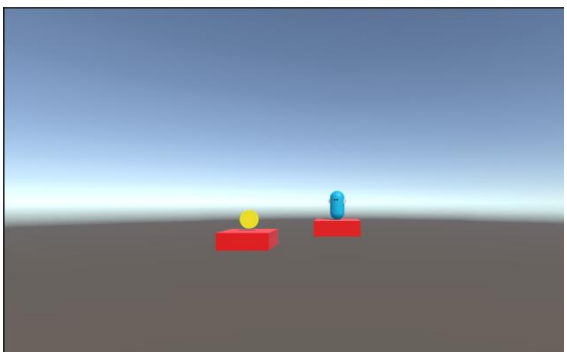

Figure 1: Simulation in Unity

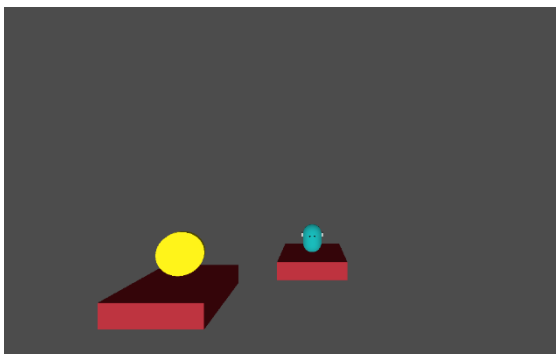

Figure 2: Simulation in Godot

The two metrics, learning- and computational runtime performance, were chosen as the learning performance is an important measure on how efficiently an agent performs a certain task whereas the runtime performance measures how long it takes an agent to learn a desired behaviour. This metric is crucial for researchers as a reduced wall-clock runtime allows for quicker iterations during hyperparameter search and enables cost reduction for server expenses in cloud-based trainings.

Experiment

According to [17], it is crucial to have standardized, reproducible and meaningful reporting methods for the evaluation of deep reinforcement learning algorithms. Reproducibility should help to reduce misinterpretation and minimize wasted efforts originating from reproducing those results. There are

extrinsic factors and intrinsic factors which are affecting reproducibility [17]. Extrinsic factors are elements like hyperparameters and the underlying codebase, whereas intrinsic factors are things like random seeds or properties of the simulation/environment.

To handle the extrinsic factors, a trial for each framework is executed with 2 different configurations of hyperparameters: the frameworks' default configuration and a configuration with shared hyperparameters. Similar to [17] each trial consists of 5 consecutive runs to ensure that results are not just due to randomness. Furthermore, each run is executed for 22 minutes, as this is the duration, where most of the learning seemed to have already happened (Figure 3). In order to tackle the intrinsic factors, each run of a trial runs with a different seed and the simulation was provided with a fixed seed.

The 5 runs of a trial are then averaged and the learning performance, the runtime performance and the corresponding confidence interval of each framework are plotted in the form of graphs to be able to easily compare the differences between the frameworks. The exact hyperparameter configurations, scripts, trial run commands, collected data, and simulation executables are provided in the form of GitHub links in the chapter "Data and Software Availability", so other researchers are able to reproduce the results.

Default Configuration

Unity ML Agents and Godot RL Agents and its respective RL backends provide a default configuration of hyperparameters, so users and researchers can easily get started with the training of RL agents. Some of these default hyperparameters vary a lot between the frameworks and thereby potentially affect the initial learning and runtime performance of the agents. This part of the experiment compares how well the frameworks perform under their default hyperparameter configuration.

Shared Configuration

As a following step, a predefined set of hyperparameters is adjusted to be the same across the different frameworks to be able to compare the performance of the different frameworks in a potentially fair manner. The adjusted hyperparameters are displayed in Table 5. Other adjustable hyperparameters are not taken further into account, as they are not available in every framework or exceed the scope of this work. Furthermore, the hyperparameters which seemed to have the most impact during hyperparameter search were buffer size, mini batch size and number of steps.

Results

Default Configuration

Table 1 displays the mean reward and 95% confidence interval for each framework after around 2, 12 and 22 minutes of training under the default hyperparameter configuration. The figures display the mean reward (averaged over 5 runs) accumulated by each framework during 22 minutes of training. Since one run of Unity ML Agents did not learn the optimal policy but got stuck in a local optimum, the confidence interval is very high as can be seen in Figure 4. This high confidence interval overlaps with all the other learning curves, making the graph harder to read and compare. For that reason, Figure 3 was introduced, which presents the performance of the different frameworks without any confidence interval, making the graph easier to read.

Duration	Godot[CleanRL]	Godot[RLlib]	Godot[SB3]	Godot[SF]	Unity ML
2m	584 (54, 1115)	22 (11, 33)	190 (85, 295)	2946 (1668, 4224)	64 (-22, 150))
12m	2474 (569, 4379	27 (14, 41)	3090 (978, 5202)	8827 (5739, 11914)	8510 (-1519, 18539)
22m	3527 (-1048, 8103)	32 (0, 65)	3558 (1402, 5714)	10383 (9661, 11106)	9975 (-1854, 21803)

Table 1: Default Config: Mean reward + 95% confidence interval after around 2, 12 & 22 minutes of training

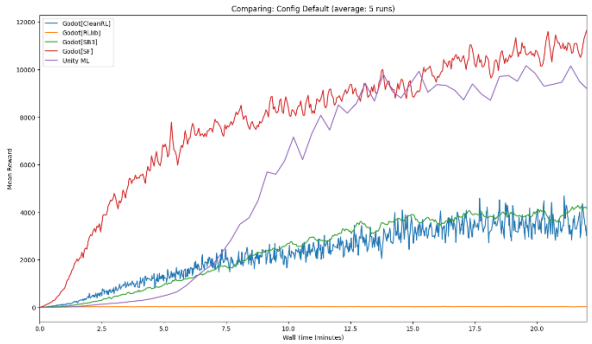

Figure 3: Comparing frameworks | Default config | Without confidence intervals

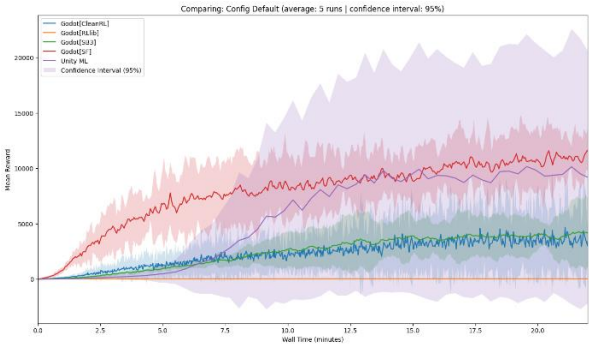

Figure 4: Comparing frameworks | Default config | With confidence intervals

Shared Configuration

Table 2 displays the mean reward and 95% confidence interval for each framework after around 2, 12 and 22 minutes of training under the shared hyperparameter configuration. The graphs display the mean reward accumulated by each framework during 22 minutes of training. Figure 5 and 6 compare the averaged results of all the different frameworks without and with confidence intervals.

Duration	Godot[CleanRL]	Godot[RLlib]	Godot[SB3]	Godot[SF]	Unity ML
2m	114 (15, 214)	5 (1, 10)	142 (94, 190)	169 (103, 235)	35 (7, 63)
12m	1787 (492, 3081)	126 (26, 226)	3846 (1977, 5715)	2675 (1476, 3875)	10042 (5224, 14859)
22m	3353 (1765, 4941)	1163 (308, 2017)	6007 (4561, 7453)	3935 (1675, 6195)	15488 (14289, 16687)

Table 2: Shared config: Mean reward + 95% confidence interval after around 2, 12 & 22 minutes of training

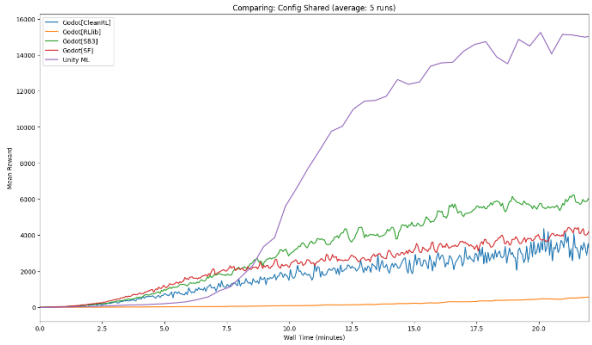

Figure 5: Comparing frameworks | Shared config | Without confidence intervals

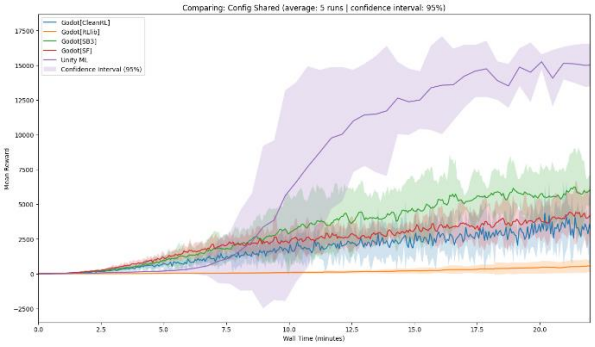

Figure 6: Comparing frameworks | Shared config | With confidence intervals

Discussion

Default Configuration

Running the trials under the frameworks' default hyperparameter configuration revealed that Unity ML Agents and Godot RL Agents utilizing Sample Factory (Godot RLAgents[SF]) performed the best

among all other frameworks (Figure 3). Both Unity ML Agents and Godot RL Agents[SF] achieve a mean reward of around 10.000 after 22 minutes of training. It is notable that Godot RL Agents[SF] manages to learn a lot faster than Unity ML Agents and all the other frameworks. After around 2 minutes, it already managed to gather a mean reward of 3.000 while Unity ML Agents is at a mean reward of 64, making Godot RL Agents[SF], at that point in time, 46 times faster than Unity ML Agents. After around 14 minutes, Unity ML Agents and Godot RL Agents[SF] achieve the same accumulated reward of around 10.000.

It is noteworthy that even though some individual runs of Unity ML Agents achieved slightly higher rewards than Godot RL Agents[SF] (Figure 7) - one run of Unity ML Agents (Run 04) did not manage to learn the optimal policy but rather got stuck in a local optimum. When reviewing the learned behavior of Run 04 via inference in the Unity game engine, it shows that the agent has not learned to use its sensors properly, but it would rather jump backwards into roughly the direction of the next platform, "hoping" that it will land. This might indicate that the performance of Unity ML Agents under the default hyperparameter configuration is more sensible to factors like random weight initialization. The failed run is also responsible for the high confidence interval for the Unity ML Agents trial, indicating that more runs are necessary to confidently measure the real performance of Unity ML Agents under the default hyperparameter configuration.

Godot RL Agents[RLlib] under the default hyperparameter configuration does not learn the intended behavior. It accumulates in 22 minutes of training a mean reward of 30, indicating that the configuration of hyperparameters needs some adaptation to achieve better results. Godot RL Agents[CleanRL] and Godot RL Agents[SB3] manage a mean reward of around 3.500.

Shared Configuration

The results of the trials under the shared hyperparameter configuration reveal that similar hyperparameters do not achieve similar performance across the frameworks. Unity ML Agents performs well under this configuration, achieving an accumulated reward of around 15.000 and thereby increasing its mean reward by 5.000. It is also notable that during the runs under the shared configuration, Unity ML Agents did not get stuck in a local optimum but always learned an optimal policy.

While Unity ML Agents' learning performance increased, Godot RL Agents [SF] only achieved an average reward of 4.000, thereby decreasing its mean reward by 6.000. This suggests that Unity ML Agents is more robust to changes in the hyperparameters than Godot RL Agents[SF], but further experiments are needed.

Finally, the learning performance of all the other Godot RL Agents frameworks improved slightly or stayed roughly the same under the shared configuration. Godot RL Agents[SB3] accumulated a reward of 6.000, making it perform better than Godot RL Agents[SF]. Godot RL Agent[CleanRL] achieved a mean reward of around 3.300. Even though Godot RL Agents[RLlib] accumulated less return compared to the other frameworks, it still managed to achieve a total return of around 1.200, making it perform 36 times better than under the default configuration.

Additional Findings

A noteworthy discovery while conducting the experiments is that the frameworks completed different numbers of total time-steps for the same amount of time (Figure 9). As an example: while it took Godot RL Agents[RLlib] around 7 minutes to complete 1 million time-steps, it took Godot RL Agents[SF] only 80 seconds. This is relevant in that it shows that the default stopping condition for

such experiments, which is often a fixed number of time-steps, does not allow for a fair comparison as can be seen in Figure 10. Looking at Figure 10, Godot RL Agents[SB3] seems to perform much better than most of the other frameworks, even though it took Godot RL Agents[SB3] around 10 minutes to reach 1 million time-steps, while it took Godot RL Agents[SF] only 80 seconds. When looking at Figure 3, it becomes apparent that Godot RL Agents[SF] has reached a mean return of 8.000 after 10 minutes of training, making it actually perform better than Godot RL Agents[SB3] under the default configuration.

Future Work

Even though a lot of care was taken into account to guarantee comparability between Unity ML Agents and Godot RL Agents and its respective RL backends, there are still some open points which can be addressed in future research. On the one hand, a more thorough search of the hyperparameter space for each individual framework, resulting in a custom-fitted hyperparameter configuration, might reveal more comparable results in terms of total return. On the other hand, regarding the shared hyperparameter configuration, further experiments could be run, where every possible hyperparameter is adjusted to be the same, to make a confident claim about the effect of same hyperparameter configurations. Furthermore, it would be interesting to conduct more experiments in different environments to see if the frameworks still achieve the same results in terms of their current ranking. Moreover, trials under different RL algorithms might reveal more differences. Finally, more trials could be run with longer duration to see if training continues to improve and to get more information on the high variance of the Unity ML Agents trials under the default configuration.

Conclusion

The trials under the default hyperparameter configuration revealed that Godot RL Agents[SF] and Unity ML Agents achieved the most return, but Godot RL Agents[SF] learned up to 46 times faster than Unity ML Agents in the first minutes of training. The trials under the shared hyperparameter configuration showed, that Godot RL Agents[SF] trained much worse than before, indicating its sensitivity to hyperparameter changes, while Unity ML Agents performed even better than previously. The other backends (CleanRL, SB3) performed reasonably well under both configurations, except for RLlib which was not able to learn at all (default configuration) or achieved relatively small rewards (shared configuration). Furthermore, the results indicate the importance of custom fitted hyperparameter configurations as implementation details seem to differ a lot. While the experiments of the different frameworks revealed big differences in the total accumulated reward, the important factor, in the end, is how well the agent behaves in the simulation or in the real world. When evaluating those results within the simulation and actually watching the agents solve the problem, it becomes apparent that the agents perform really well regardless if they accumulate a mean reward of 10.0000 or 5.000 and therefore most of the frameworks are suitable for achieving the desired goal in the present reinforcement problem. However, in terms of quick iterations, short wall clock runtime and reduced costs, Godot RL Agents[SF] is the clear winner among those frameworks. Unity ML Agents on the other hand worked considerably well under various hyperparameter configurations, indicating its robustness.

Competing Interests Statement

This work compares Unity ML Agents with Godot RL Agents. The first author of this work has contributed to the development of Godot RL Agents.

Data and Software Availability

Source Code & Builds

This section includes the links to the source code of the simulations in Unity (1) and Godot (2). Furthermore, it also contains the builds and individual hyperparameter configurations for the trials in Unity ML Agents (3) and Godot RL Agents (4).

1. https://github.com/visuallization/unity-ml-platformer
2. https://github.com/visuallization/godot-4-ml-platformer
3. https://github.com/visuallization/unity_ml_platformer_experiments
4. https://github.com/visuallization/godot_4_ml_platformer_experiments

Data & Graphs

This section contains the link to the data of the experiments and the source code of the visualizations.

1. https://github.com/visuallization/rl_master_thesis_evaluation

References

[1] OpenAI, 'GPT-4 Technical Report'. arXiv, Mar. 27, 2023. Accessed: Jun. 18, 2023. [Online]. Available: http://arxiv.org/abs/2303.08774

[2] M. Bojarski *et al.*, 'End to End Learning for Self-Driving Cars'. arXiv, Apr. 25, 2016. doi: 10.48550/arXiv.1604.07316.

[3] J. Jumper *et al.*, 'Highly accurate protein structure prediction with AlphaFold', *Nature*, vol. 596, no. 7873, Art. no. 7873, Aug. 2021, doi: 10.1038/s41586-021-03819-2.

[4] V. Mnih *et al.*, 'Playing Atari with Deep Reinforcement Learning'. arXiv, Dec. 19, 2013. Accessed: Sep. 06, 2023. [Online]. Available: http://arxiv.org/abs/1312.5602

[5] D. Silver *et al.*, 'Mastering the game of Go without human knowledge', *Nature*, vol. 550, no. 7676, Art. no. 7676, Oct. 2017, doi: 10.1038/nature24270.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. in Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.

[7] Y. Liu *et al.*, 'Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models'. arXiv, May 10, 2023. Accessed: Jun. 18, 2023. [Online]. Available: http://arxiv.org/abs/2304.01852

[8] D. Silver *et al.*, 'A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play', *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018, doi: 10.1126/science.aar6404.

[9] S. Huang, R. F. J. Dossa, C. Ye, and J. Braga, 'CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms'. arXiv, Nov. 16, 2021. Accessed: Jun. 16, 2023. [Online]. Available: http://arxiv.org/abs/2111.08819

[10] E. Liang *et al.*, 'RLlib: Abstractions for Distributed Reinforcement Learning'. arXiv, Jun. 28, 2018. Accessed: Jun. 16, 2023. [Online]. Available: http://arxiv.org/abs/1712.09381

[11] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, 'Stable-Baselines3: Reliable Reinforcement Learning Implementations', *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, 2021.

[12] A. Petrenko, Z. Huang, T. Kumar, G. Sukhatme, and V. Koltun, 'Sample Factory: Egocentric 3D Control from Pixels at 100000 FPS with Asynchronous Reinforcement Learning'. arXiv, Jun. 22, 2020. Accessed: Jun. 16, 2023. [Online]. Available: http://arxiv.org/abs/2006.11751

[13] A. Juliani *et al.*, 'Unity: A General Platform for Intelligent Agents'. arXiv, May 06, 2020. doi: 10.48550/arXiv.1809.02627.

[14] E. Beeching, J. Debangoye, O. Simonin, and C. Wolf, 'Godot Reinforcement Learning Agents'. arXiv, Dec. 07, 2021. doi: 10.48550/arXiv.2112.03636.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, 'Proximal Policy Optimization Algorithms'. arXiv, Aug. 28, 2017. Accessed: Jun. 17, 2023. [Online]. Available: http://arxiv.org/abs/1707.06347

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, 'Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor'. arXiv, Aug. 08, 2018. Accessed: Jun. 17, 2023. [Online]. Available: http://arxiv.org/abs/1801.01290

[17] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, 'Deep Reinforcement Learning that Matters'. arXiv, Jan. 29, 2019. doi: 10.48550/arXiv.1709.06560.

Supplementary material

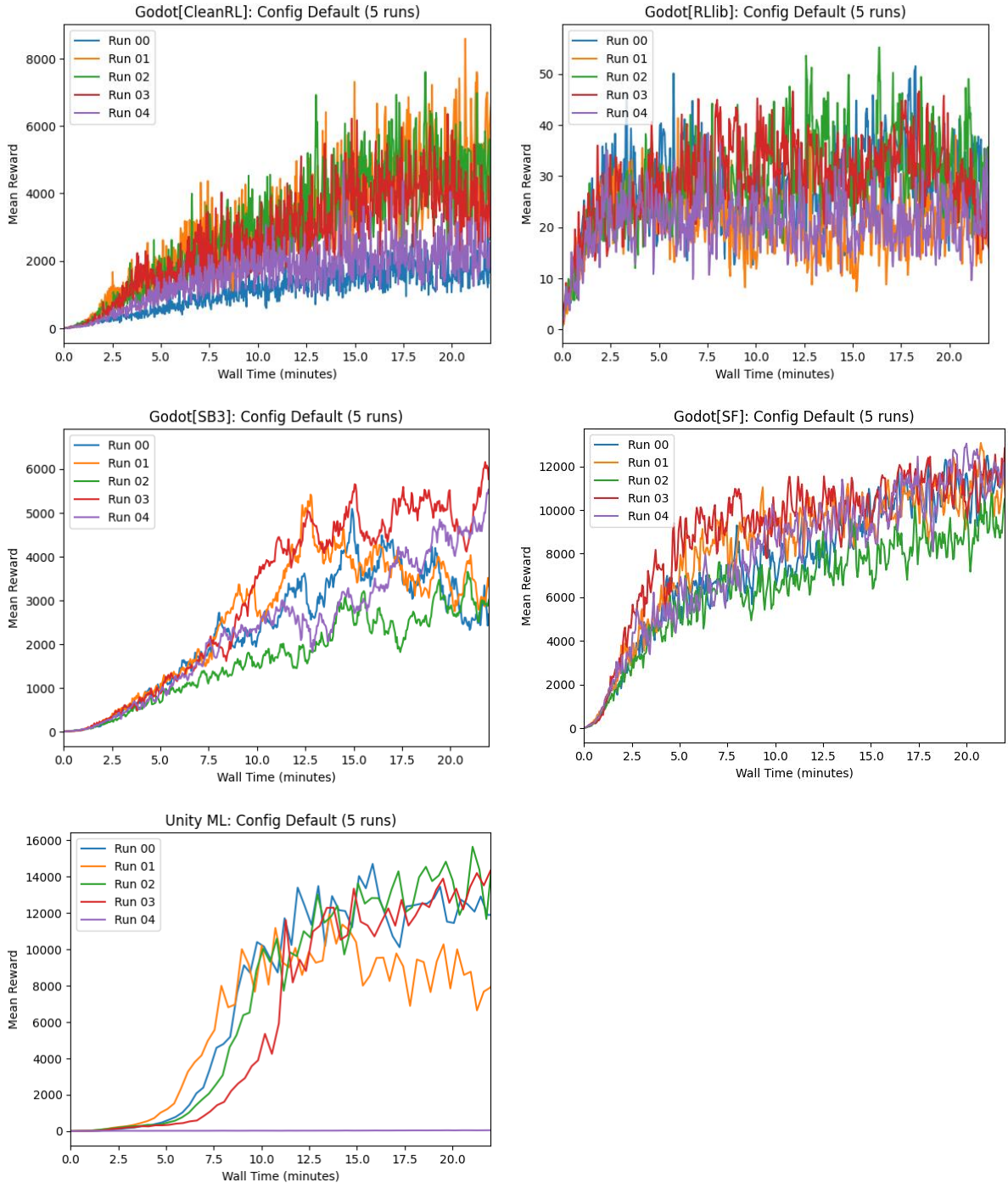

Figure 7: Individual runs | Default configuration

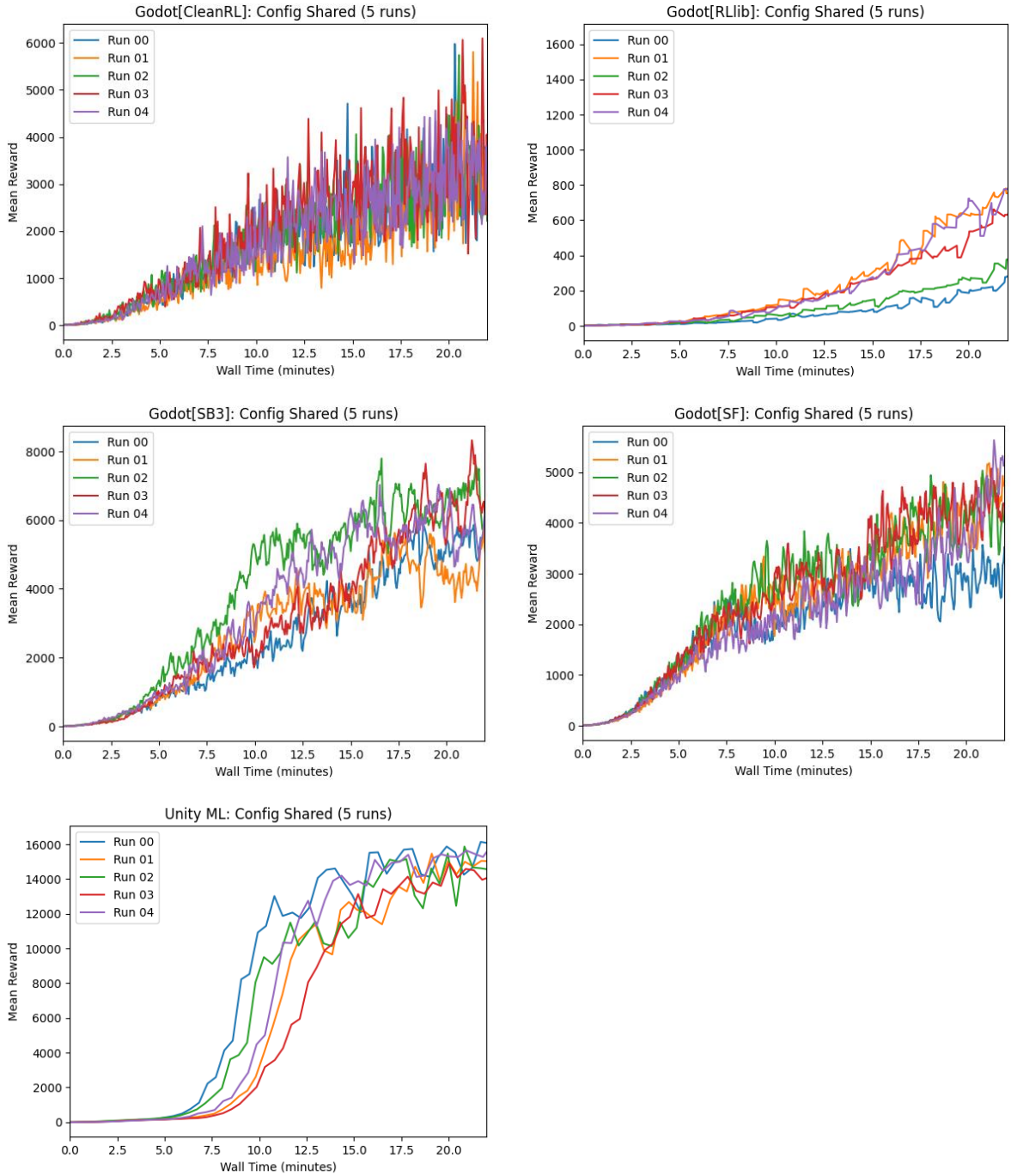

Figure 8: Individual runs | Shared configuration

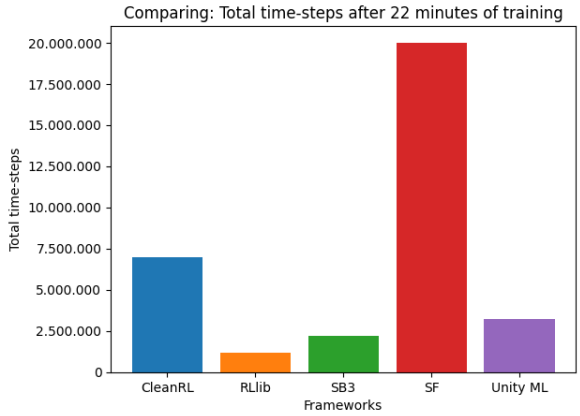

Figure 9: Comparing total time-steps after 22 minutes of training

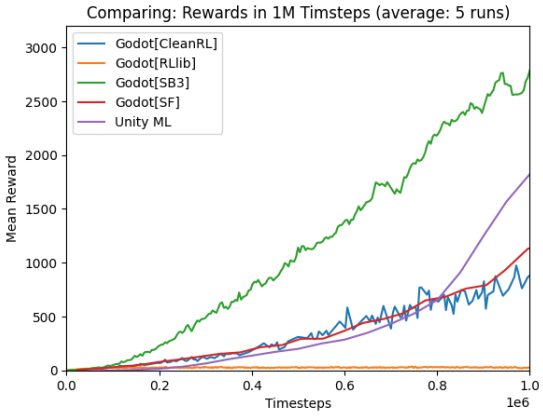

Figure 10: Comparing rewards after 1 million time-steps

Godot[CleanRL]	Godot[RLlib]	Godot[SB3]	Godot[SF]	Unity ML
896 (-158, 1950)	26 (18, 34)	2777 (1052, 4502)	1131 (743, 1519)	1823 (-1349, 4994)

Table**3**: Default config: Mean reward + 95% confidence interval after 1 million time-steps

Godot[CleanRL]	Godot[RLlib]	Godot[SB3]	Godot[SF]	Unity ML
total-timesteps	timesteps_total	total_timesteps	train_for_env_steps	max_steps
learning-rate	lr	learning_rate	learning_rate	learning_rate
anneal-lr	lr_schedule	lr_schedule	lr_schedule	learning_rate_schedule
clip-coef	clip_param	clip_range	ppo_clip_ratio	epsilon
num-envs * num-steps	train_batch_size	n_envs * n_steps	num_batches_per_epoch * batch_size	buffer_size
(num-envs * num-steps) / num-minibatches	sgd_minibatch_size	batch_size	batch_size	batch_size
gamma	gamma	gamma	gamma	gamma
gae-lambda	lambda	gae_lambda	gae_lambda	lambd
update-epochs	num_sgd_iter	n_epochs	num_epochs	num_epoch
num-steps	horizon	n_steps	rollout	time_horizon
ent-coef	entropy_coeff	ent_coef	exploration_loss_coeff	beta

Table 4: Names of hyperparameters in Unity ML Agents & Godot RL Agents

Buffer Size	Mini Batch Size	Steps	Epochs	Learning Rate	Beta	Gamma	Epsilon
16384	128	128	3	0.0003	0.005	0.99	0.2

Table 5: Shared Config - Adjusted hyperparameters

Lenovo ThinkBook 16p Gen 2

The notebook contains an AMD® Ryzen 9 5900HX (8 cores, 16 threads) central processing units (CPUs), 32 gigabyte (GB) random access memory (RAM) and a 1 terabyte (TB) solid-state-drive (SSD). Additionally, it features two different graphics processing unit (GPU), a Radeon Graphics (8 cores) and a NVIDIA GeForce RTX 3060 (6GB video random access memory (VRAM)).

MDP

The actions, observations and rewards of this work's MDP are described as follows.

Actions

The agent is able to perform the actions move (forward, backward), rotate (left, right) and jump (if it is currently touching the floor of a platform).

Observations

The agent observes its environment via a ray cast sensor, which is providing it with information about the existence and distance of objects in front of it. Furthermore, the agent can observe it's distance and direction to the goal and if it is currently touching the floor.

Rewards

The agent gets a positive reward of 100 every time it collects a coin and a small positive reward when it is moving into the direction of the next goal platform. Furthermore, it gets a negative reward of -0.01 for every passed time-step.