High Performance Simulation for Scalable Multi-Agent Reinforcement Learning

Anonymous Authors[1]

Abstract

Multi-agent reinforcement learning experiments and open-source training environments are typically limited in scale, supporting tens or sometimes up to hundreds of interacting agents. In this paper we demonstrate the use of *Vogue*, a high performance agent based model (ABM) framework. Vogue serves as a multi-agent training environment, supporting thousands to tens of thousands of interacting agents while maintaining high training throughput by running both the environment and reinforcement learning (RL) agents on the GPU. High performance multi-agent environments at this scale have the potential to enable the learning of robust and flexible policies for use in ABMs and simulations of complex systems. We demonstrate training performance with two newly developed, large scale multi-agent training environments. Moreover, we show that these environments can train shared RL policies on time-scales of minutes and hours.

1. Introduction

Increases in computational power and available data have led to an increase in the potential scale and fidelity of computer simulations such as agent based models (ABMs). In tandem, a desire has developed for complex agent behaviour which cannot be addressed solely by the often error-prone and challenging process of manually designing and implementing agent behaviours that are both flexible and robust to extreme states of the simulation.

In the context of agent-based models, multi-agent reinforcement learning (MARL) is a means of learning agent behaviours where the reward function is well known but the simulation has a complex state space. This motivates the development of fast and large-scale training environments, enabling the use of MARL to learn behaviours reflecting emergent systems such as crowds, flocks, social networks or markets.

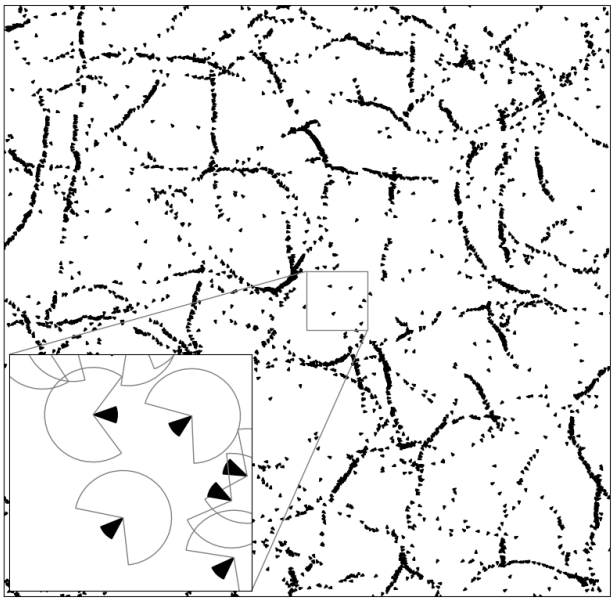


Figure 1. Snapshot of agents in the *flock* MARL environment. The agents are rewarded for being close to other agents, but are penalised for collisions. This example contains 5,000 agents learning a shared policy. 500 training steps (totalling 625,000 mini batches) took approximately 16 minutes to train on a laptop Nvidia GTX 1650 GPU. The inset illustrates how agents sense neighbouring agents via a simple vision model, shown in more detail in Figure 2(a).

2. High Performance MARL with Vogue

2.1. Vogue: An Agent-Based Modelling Framework

Vogue is a commercial Python framework for high performance ABMs. It is motivated by some key design principles:

- **Usability:** Vogue allows users to write ABM logic in native, idiomatic Python, as shown in code snippet 1.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

This is then compiled by Vogue into high performance bytecode.

- **High Performance Execution:** Vogue maps model logic over agents using high performance, vectorised update patterns. This is efficient, and obviates the need for modellers to consider the computational complexity and low-level implementation details of ABMs.
- **Composition:** Vogue model logic is described in small pieces of functionality following standard patterns. This allows modellers to easily combine and reuse model logic, and also standardise how models are written.
- **Interoperability:** Vogue interacts seamlessly with the broader Python data science and machine learning (ML) ecosystem. This allows Vogue models to be easily employed in ML and data science pipelines.

These design choices allow users to quickly implement high performance ABMs while retaining a high degree of abstraction in the implementation, and deploy Vogue alongside popular Python ML tools.

```
@vogue.cuda.interaction.self(
    BoidParams, Boid
)
def update_position(params, boid):
    boid.x = (boid.x + boid.vx) % 1.0
    boid.y = (boid.y + boid.vy) % 1.0
```

Code Snippet 1. A Vogue interaction updating the positions of agents in the flock multi-agent environment. Vogue allows ABM logic to be written in idiomatic native python, with the processing of updates and execution of the model handled by the Vogue engine.

2.2. End-to-end GPU-based Reinforcement Learning

A major advantage of Vogue is that models can be executed completely on the GPU with minimal changes to the underlying code, leveraging the high degree of parallelism afforded by the GPU to efficiently update the state of the model. Vogue's implementation allows data to be shared between Vogue and other ML and deep-learning libraries without copying (or transfer via the host) via the CUDA array interface. This means Vogue can interact directly with any ML frameworks that implement the CUDA array interface, including popular ML tools like PyTorch (Paszke et al., 2019) and JAX (Bradbury et al., 2018), or GPU based data tools like RAPIDS (Team, 2018). This allows the execution of both the Vogue training environment/simulation and the deep reinforcement learning (RL) agent on the GPU, removing the overhead of moving data to and from the GPU during training.

3. Background

MARL has found a wide range of applications including distributed autonomous systems, socioeconomic and game theoretic problems (Busoniu et al., 2008). The number of learning agents in these systems typically ranges from single-digit to hundreds of interacting agents.

A major obstacle to applying MARL to systems with larger numbers of interacting agents is the lack of high performance simulation environments, as well as the technical challenges and cost associated with their development. Training environments are often implemented for execution on the CPU with the deep RL agent running on the GPU. However, the communication of data from the simulation on the host to the agent on the GPU can have a significant performance impact.

OpenAI Gym (Brockman et al., 2016) has become a de facto standard API for RL research, allowing training environments and RL agent implementations to be easily combined, though it does not have a standard API for multi-agent training. There are a number of open-source MARL environment projects implementing this API, including Petting Zoo (Terry et al., 2020) and MAgent (Zheng et al., 2018). MAgent potentially supports environments containing millions of agents, but in contrast to Vogue (proposed here) the environments restrict agents to a discrete grid and the engine is implemented in C++. Petting Zoo wraps a number of environments (including MAgent), though many of the environments are restricted to tens or hundreds of agents, or have observation spaces that scale with the number of agents, restricting their usable scale.

RLLib (Liang et al., 2017) provides a set of tools for the execution of RL pipelines, including automated hyperparameter tuning and distributed training. It has the ability to run pipelines with multiple multi-agent policies, and multiple environments executed in parallel. It does however rely on users to implement the training environment (or providing a pre-existing environment).

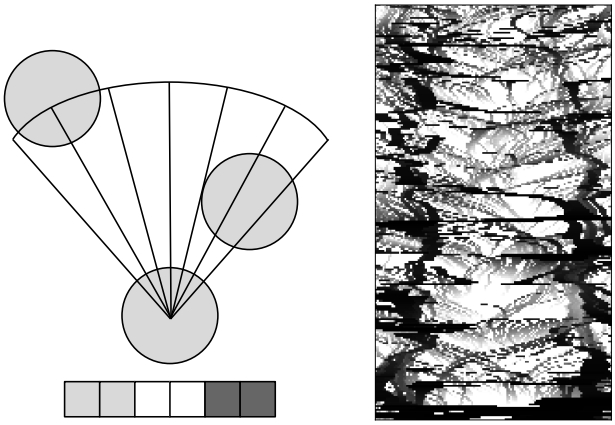
Warpdrive (Lan et al., 2021) offers a framework for end-to-end MARL on GPU. It demonstrates high training throughput, though is only tested (in Lan et al., 2021) on a maximum of 1,000 agents (executed across 2,000 environments simultaneously). Its end-to-end solution contrasts Vogue's focus on ease of writing high performance models, and interoperability with existing Python tools.

There are a number of ABM frameworks, sharing the broad aim of allowing users to more easily implement ABMs, or provide performance gains. Mesa (Kazil et al., 2020) is a popular Python ABM framework, though its performance is restricted by Python's native performance. Netlogo (Wilensky, 1999) is instead implemented in Java, though this limits its interoperability with Python and requires users to imple-

ment models in Netlogo's scripting language. Few frameworks support GPU execution; Flame (Richmond et al., 2021) is a notable example, implemented in C++ with an optional Python interface.

4. Training Environments

At the time of writing, we could not locate any standardised examples of training environments at the scale of 1,000+ interacting agents. Instead, we provide two simple examples by implementing extended versions of some popular models: *flocking* and *tag*. We note that extending the range of example environments is a priority for future work.



(a) Agent vision model (b) Agent's view over time

Figure 2. Agents visualise their local environment using a simple ray-casting model, segmenting their field of vision into a fixed number of cells (a). The resulting array (shown below the agent) represents the distance to the nearest neighbouring agent in the simulation. An example time series of an agent's view is shown in image (b); darker areas are objects closer to agent.

In both examples, to address the increased overall size of the simulation state space, the agents only observe their local neighbourhood of the simulation via a simple ray casting algorithm that divides an agent's field of view into a fixed number, v, of sectors. Internally, their view is represented by an array of v values representing the distance to the nearest object at that angle, as shown in Figure 2(a). An example time series showing an agent's view during a training step is shown in Figure 2(b). This local view forms part of the training environment observation space, local to each agent in the environment.

4.1. Flocking

This training environment is based on Reynolds' Boids model (Reynolds, 1987), designed to mimic natural flocks and shoals. The training environment consists of a flock of agents navigating a 2d space, with no hard-coded navigation rules. Previous studies have looked at emergence of flocking from reinforcement learning (for example Durve et al., 2020; Dong et al., 2021) but in this work we aim to examine the feasibility of efficiently training flocking behaviours for 1,000-10,000+ interacting agents. A snapshot of a simulation state is shown in Figure 1.

The observation space of the environment (for an individual agent) is a 129 length vector containing the agent's view, with $v = 128$ sectors, and the agent's current speed. The action space is two dimensional. One dimension accelerates/decelerates the agent, in the range $[-a_{\text{max}}, a_{\text{max}}]$, updating the speed of the agent as $s_{t+1} = \min(s_{\text{min}}, \max(s_t + a_t, s_{\text{max}}))$. The second dimension in the range $[-\theta_{\text{max}}, \theta_{\text{max}}]$ rotates the heading of the agents up to a maximum rotation rate. The reward signal for each agent is the sum of contributions due to proximity to other agents, i.e. the reward r_i of agent i is

$$r_i = \sum_j f(d_{ij}) \quad \text{for} \quad d_{ij} < d_v, \tag{1}$$

where d_{ij} is the Euclidean distance between agents i, j, d_v is the visual range of the agent, and f is the reward contribution, as in Figure 3.

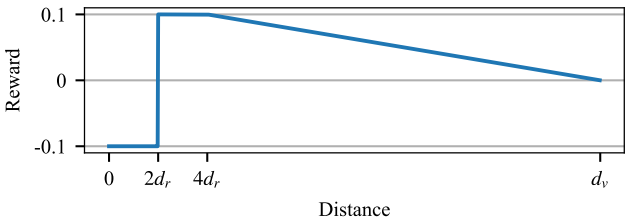


Figure 3. Flock reward function, f, where d_r is the agents' radius, and d_v the agents' vision range. Agents are penalised when they collide at a distance of $2d_r$

In each environment update, the agents are accelerated and rotated (according to actions sampled from the current policy), after which their local view model and rewards are updated for all pairs of agents in spatial proximity.

4.2. Tag

This environment is based on Petting Zoo's (in Terry et al., 2020) *simple tag* environment. It contains two agent types: *runners* and *chasers*. Chasers are rewarded for touching runners, and runners are rewarded for staying close to other runners, and penalised for touching a chaser. The view model of the agent has two simulated colour channels allowing agents to distinguish between neighbouring runners or chasers. The corresponding observation space is a 128

Algorithm 1 PPO Multi Agent Training Loop

Input: Initial policy parameters θ_0, training steps T, environment updates t, training epochs p
for $i = 0, 1, ..., T$ **do**
  Collect $n \times t$ trajectories $\mathcal{D}_i$ by running policy $\pi_i = \pi(\theta_i)$
  Calculate rewards R_i and advantage estimates A_i
  for $j = 0, 1, ..., p$ **do**
    for mini-batch b in $\mathcal{D}_i$ **do**
      $\theta_{i+1} \leftarrow$ Update_policy(θ_i, b)
    end for
  end for
end for

length vector, concatenating the two $v = 64$ colour channels. The two dimensional action space rotates the agent's heading, in the range $[-\theta_{\text{max}}, \theta_{\text{max}}]$ and moves them along the heading, in the range $[0, s_{\text{max}}]$ where s_{max} is the maximum speed of the agents.

4.3. Training

For both environments we used proximal policy optimisation (PPO) (Schulman et al., 2017) for a continuous action space, with the simulated agents sharing a single policy. In the tag environment, the runner and chaser types share independent policies, trained simultaneously. Instead of multiple parallel environments, we treat each individual agent as an independent learner, and gather trajectories across the full set of of agents. If we have n agents and run t updates of the simulation each training step, we sample a total of $n \times t$ trajectories per training step. The training process is shown in Algorithm 1.

5. Performance Results

Benchmarking was performed on an Nvidia A100 GPU. Each full training run includes $T = 200$ training steps. Inside each training step the environment was updated for $t = 128$ steps, generating $m = n \times t$ individual trajectories. The policy was then updated for $p = 2$ epochs, across b mini-batches sampled from the trajectories. The number of trajectories scales with the number of simulated agents, so the number of mini-batches is capped as $b = \min(m/|b|, b_{max})$ where $b_{max} = 512$ is the maximum number of batches, and $|b| = 512$ is the chosen mini-batch size. This means for the highest number of entities each training run performs $T \times p \times b = 204,800$ mini-batch updates of the PPO policy, and $T \times t = 25,600$ updates of the environment. The agents have a view distance equivalent to 1/10th of the width of the environment and a convex field of vision of approximately 250°. The RL agent update, and trajectory processing algorithms were implemented in JAX. The actor and critic PPO networks had two hidden layers with 64 nodes each.

It should also be noted that performance of the simulation can be highly dependent on the parameters of the simulation. For example the view distance of the agents and their vision angle impact the number of interactions between agents that must be processed.

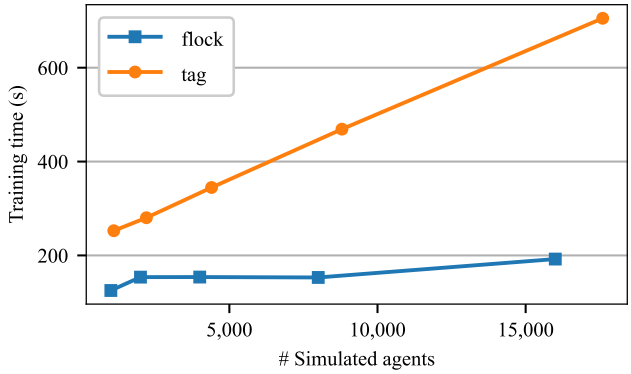


Figure 4. Total training time for 200 training steps, for the flock and tag environments, with increasing number of agents. Each training step includes running the model for 128 steps with the current policy, and updating the policy over 1024 mini-batches. Results are shown averaged over 10 training runs, though error bars are omitted due to their small relative size. For the tag environment 1/10th of the shown number of agents are chasers.

The total training time for 200 steps using the flock and tag environments are shown in Figure 4. Values shown are averaged over 10 independent training runs. With 16,000 agents, the flock environment takes around 3.5 minutes to complete training, and demonstrates slow scaling with number of agents. For a tag environment with a total of 17,600 agents (16,000 runners and 1,600 chasers) training takes around 12 minutes. The tag environment demonstrates roughly linear growth in training time with number of agents, we speculate because of the added complexity of multiple interactions between different agent types.

6. Conclusion

We have demonstrated that joining forward simulation and training of MARL systems into a single end-to-end RL process can lead to drastic improvements in performance. Using the commercial software *Vogue*, high performance ABMs can be executed directly on the GPU, while the high level of interoperability enables direct communication between Vogue and many Python based RL or deep learning frameworks. Vogue is currently available for academic research – please contact the authors.

References

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI gym, 2016.

Busoniu, L., Babuska, R., and De Schutter, B. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38:156 – 172, 04 2008. doi: 10.1109/TSMCC.2007.913919.

Dong, H., Wang, T., Liu, J., and Zhang, C. Birds of a feather flock together: A close look at cooperation emergence via multi-agent RL. *CoRR*, abs/2104.11455, 2021. URL https://arxiv.org/abs/2104.11455.

Durve, M., Peruani, F., and Celani, A. Learning to flock through reinforcement. *Physical Review E*, 102(1), jul 2020. doi: 10.1103/physreve.102.012601. URL https://doi.org/10.1103%2Fphysreve.102.012601.

Kazil, J., Masad, D., and Crooks, A. Utilizing python for agent-based modeling: The mesa framework. In Thomson, R., Bisgin, H., Dancy, C., Hyder, A., and Hussain, M. (eds.), *Social, Cultural, and Behavioral Modeling*, pp. 308–317, Cham, 2020. Springer International Publishing. ISBN 978-3-030-61255-9.

Lan, T., Srinivasa, S., and Zheng, S. Warpdrive: Extremely fast end-to-end deep multi-agent reinforcement learning on a GPU. *CoRR*, abs/2108.13976, 2021. URL https://arxiv.org/abs/2108.13976.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., and Stoica, I. Ray rllib: A composable and scalable reinforcement learning library. *CoRR*, abs/1712.09381, 2017. URL http://arxiv.org/abs/1712.09381.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pp. 25–34, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0897912276. doi: 10.1145/37401.37406. URL https://doi.org/10.1145/37401.37406.

Richmond, P., Chisholm, R., Heywood, P., Leach, M., and Kabiri Chimeh, M. Flame gpu, December 2021. URL https://doi.org/10.5281/zenodo.5769677.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Team, R. D. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018. URL https://rapids.ai.

Terry, J. K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sulivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C., Williams, N. L., Lokesh, Y., Sullivan, R., and Ravi, P. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020.

Wilensky, U. Netlogo. http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999. URL http://ccl.northwestern.edu/netlogo/.

Zheng, L., Yang, J., Cai, H., Zhou, M., Zhang, W., Wang, J., and Yu, Y. Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.