

# Optimizing Distributed Reinforcement Learning with Reactor Model and Lingua Franca

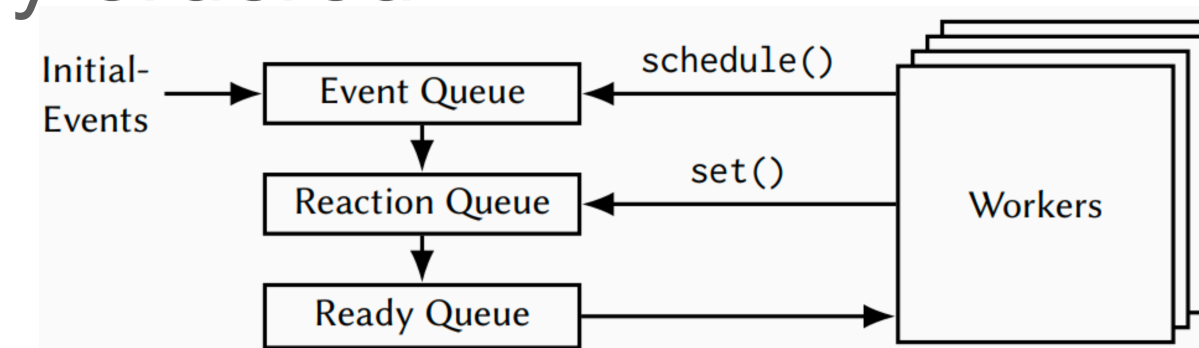
Pranav Atreya, Jacky Kwok, Amil Dravid

## Overview

- Reinforcement learning (RL) workloads come with unique considerations that common distributed computing design patterns (e.g., MapReduce) don't satisfy well
- Typical SoTA frameworks for distributed RL (e.g., Ray[1]) employ the actor model of concurrency
- Hypothesis: Can the reactor model of computation and thread-based parallelism accelerate distributed RL workloads?**

## Actor Model (Ray)

- “Actor” as primary unit of computation, communication through asynchronous message passing
- Message processing by actors need not be strictly ordered
- We use Ray’s implementation of the actor model



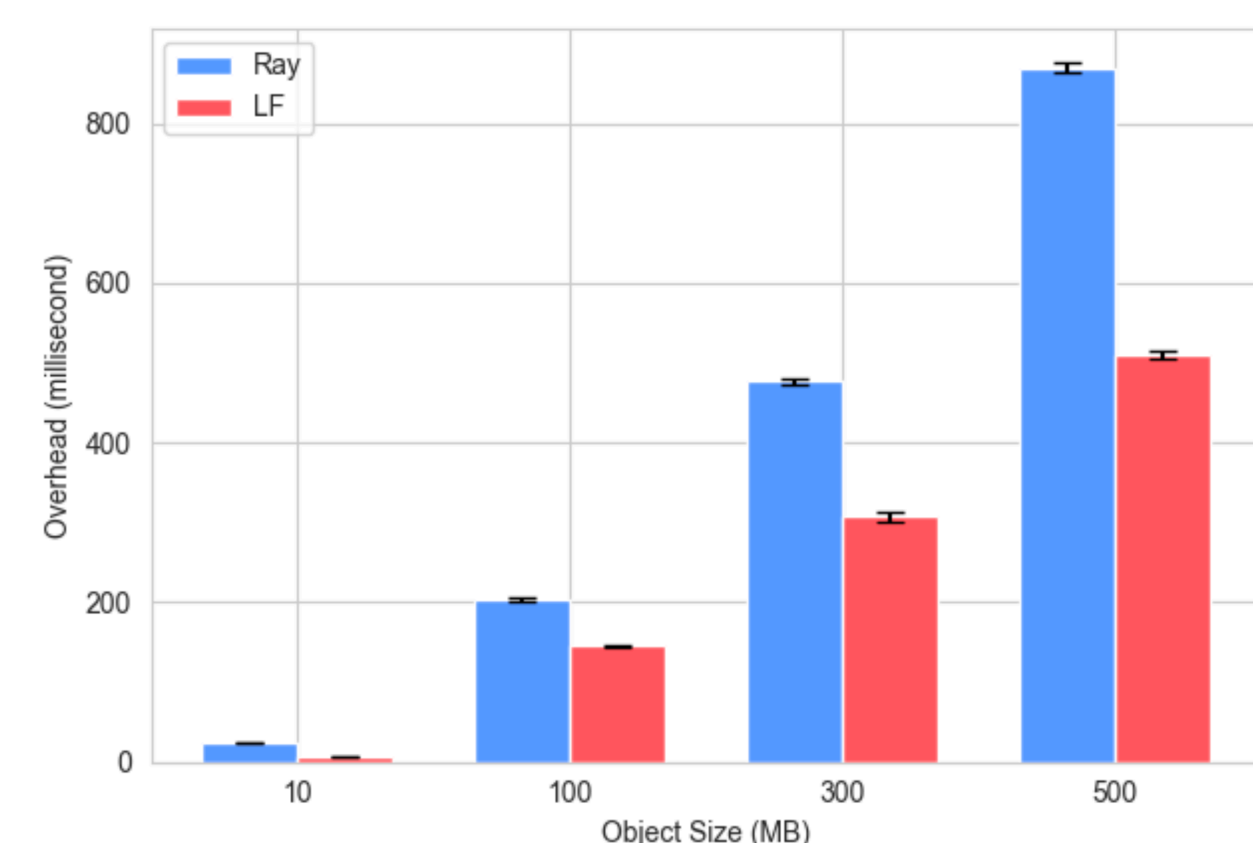
## Reactor Model (Lingua Franca)

- Enables deterministic concurrency through “reactors” and “reactions”. Instead of messages, reactors react to discrete events each linked to a logical time by triggering reactions. Can be thought of as “sparse synchronous model”.
- Reactions can modify state shared by other reactions in the same reactor, but communication across reactors is solely through events.
- Reactors don't directly reference peers, enabling hierarchy
- “Actions” bridge internal determinism with external nondeterminism
- Lingua Franca [2] (partially contributed to by us) implements the reactor model

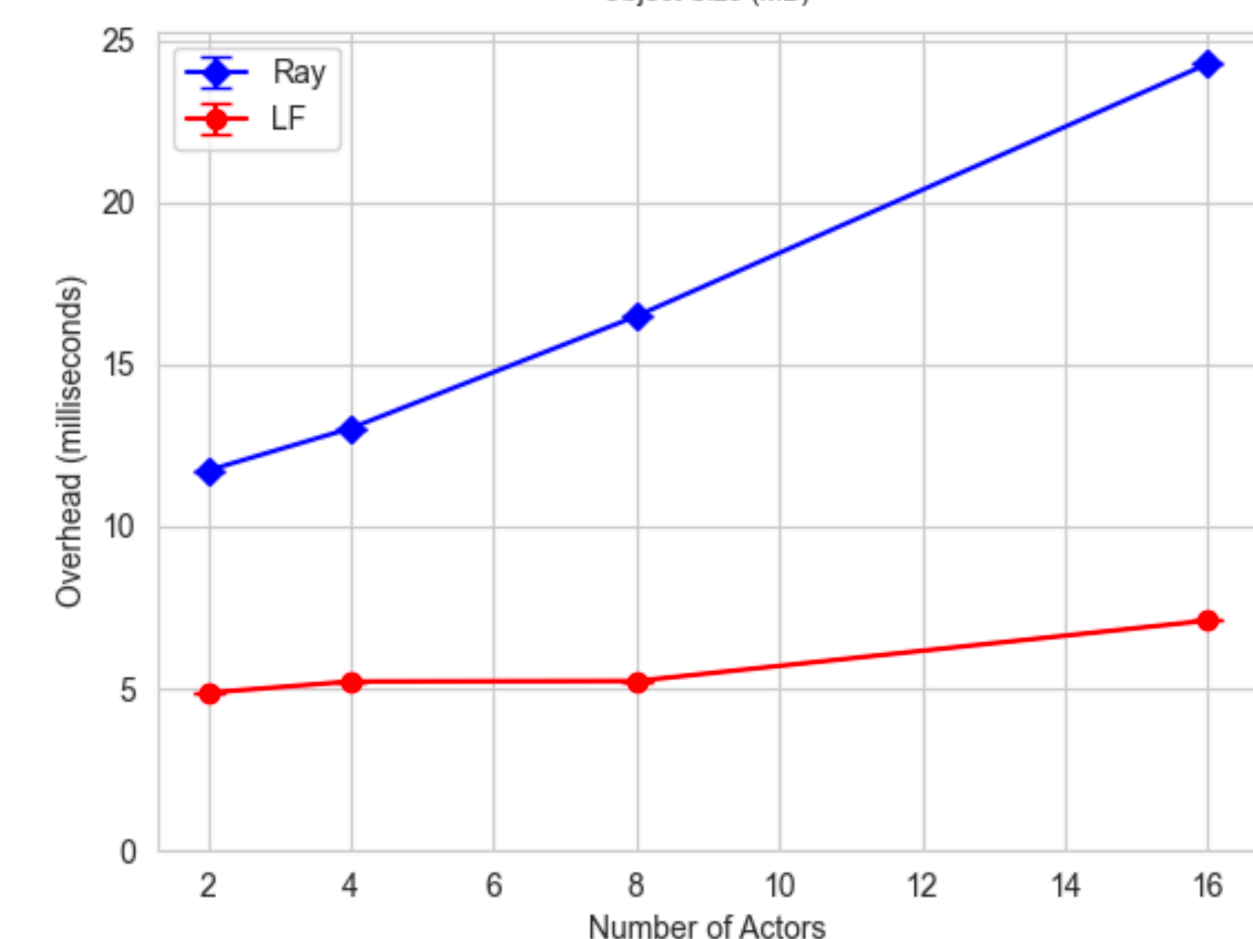
```

1 #import packages
2 #e.g. Torch, Gym, Numpy
3
4 reactor RolloutReactor {
5   input gradients
6   output trajectories
7
8   state EnvironmentState
9   state PolicyState
10  state ActionBuffer
11  state RewardBuffer
12  state ObservationBuffer
13
14  reaction(Startup) {
15    # Initialize Environment
16  }
17
18  reaction(gradients) -> trajectories {
19    # Perform rollouts for the Environment
20  }
21
22 reactor ReplayBufferReactor {
23   input [6] trajectories
24   output dataset
25
26   state ExperienceData
27   state SamplingPointer
28   state PrioritizedInfo
29
30  reaction(Startup) {
31    # Initialize ReplayBuffer
32  }
33
34  reaction(trajectories) -> gradients {
35    # Append trajectories into ReplayBuffer
36  }
37
38 reactor LearnerReactor {
39   output gradients
40   input [6] dataset
41
42   state ModelParameter
43   state OptimizerState
44   state LearningRate
45   state TargetNetworkParameters
46
47  reaction(Startup) -> gradients {
48    # Initialize the policy
49  }
50
51  reaction(dataset) -> gradients {
52    # Update the policy
53  }
54
55 # Main reactor {
56   rollout = new(6) RolloutReactor()
57   replay = new(6) ReplayBufferReactor()
58   learner = new(6) LearnerReactor()
59
60   # Specific
61   (Learner.gradients) -> rollout.gradients
62   (rollout.trajectories) -> replay.trajectories
63   (replay.dataset) -> learner.dataset
64 }
    
```

RL in LF

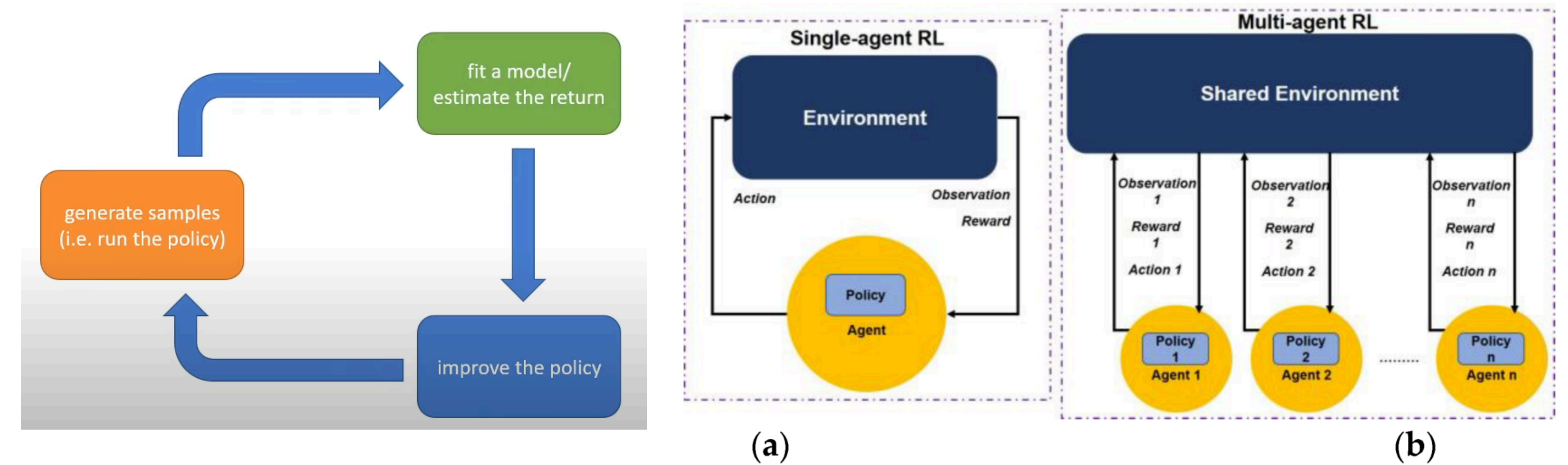


Broadcast & gather Times vs Object Size



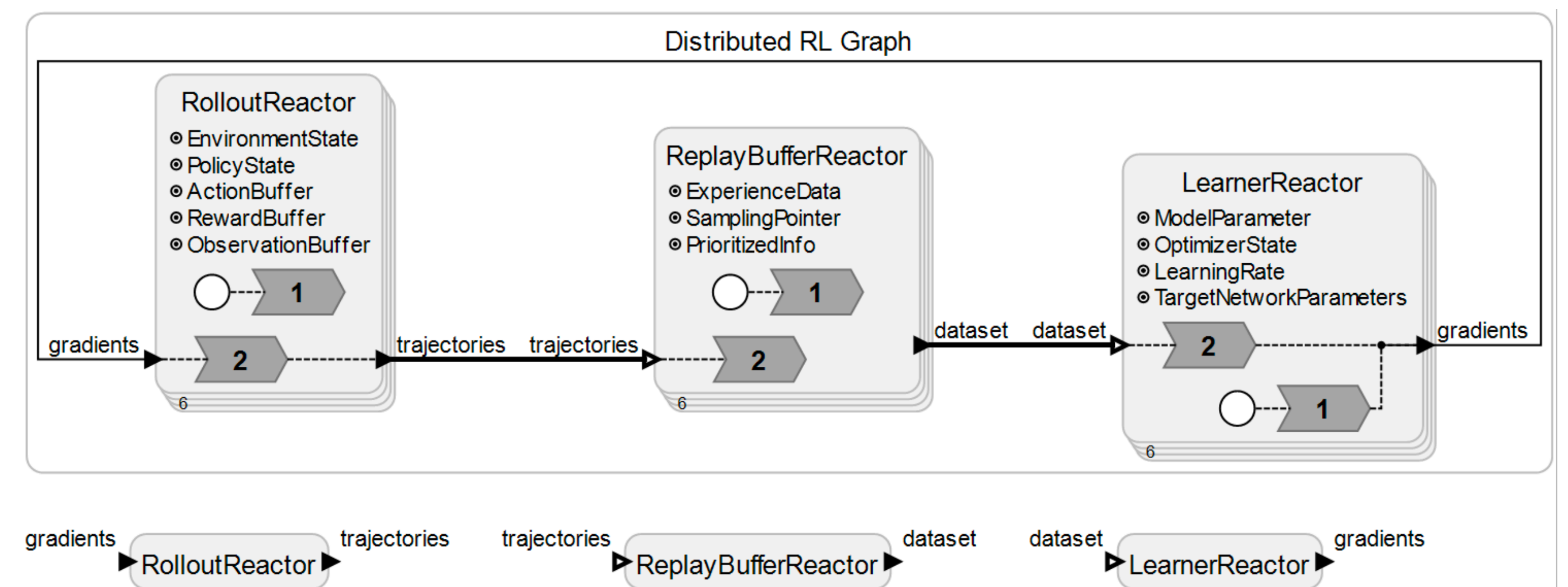
Broadcast & gather Times vs # Actors

## Distributed Reinforcement Learning



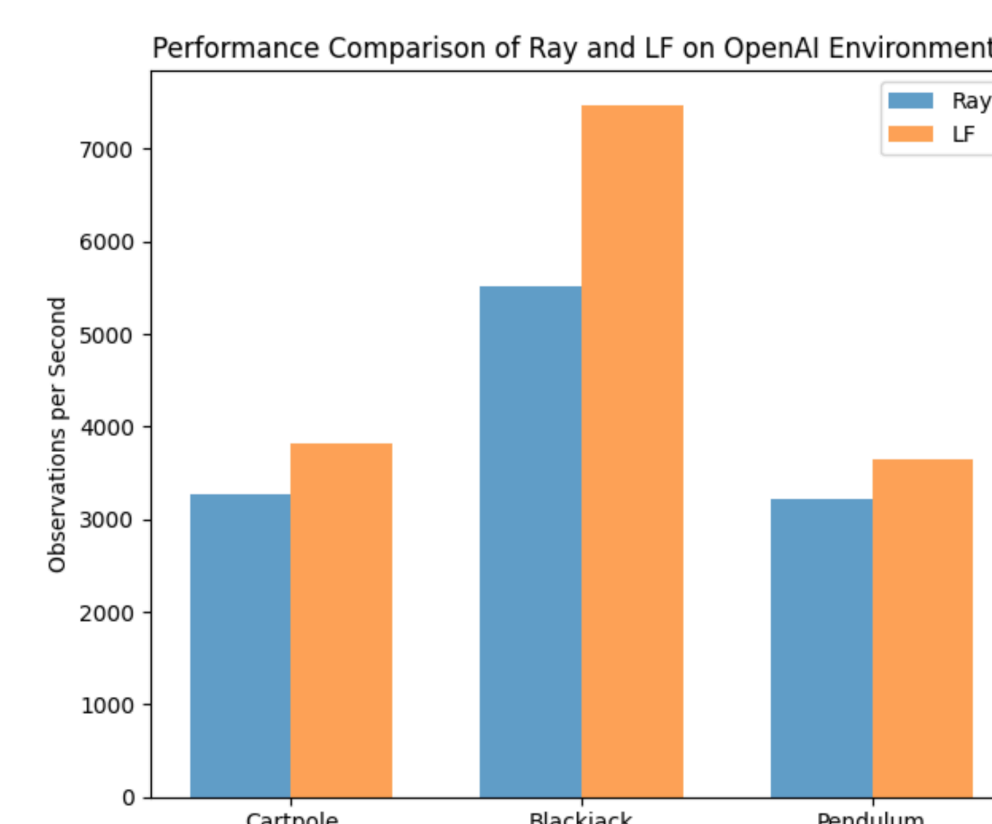
RL Pipeline

(a) Single-agent model of RL, (b) multi-agent model

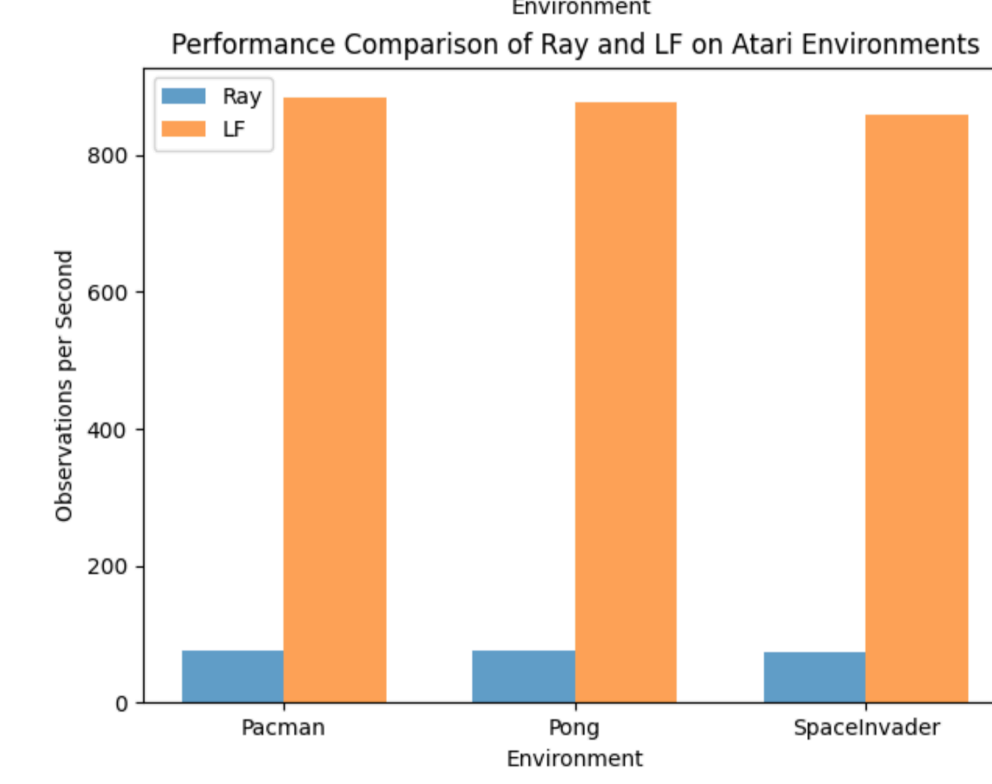


Dataflow graph for distributed RL

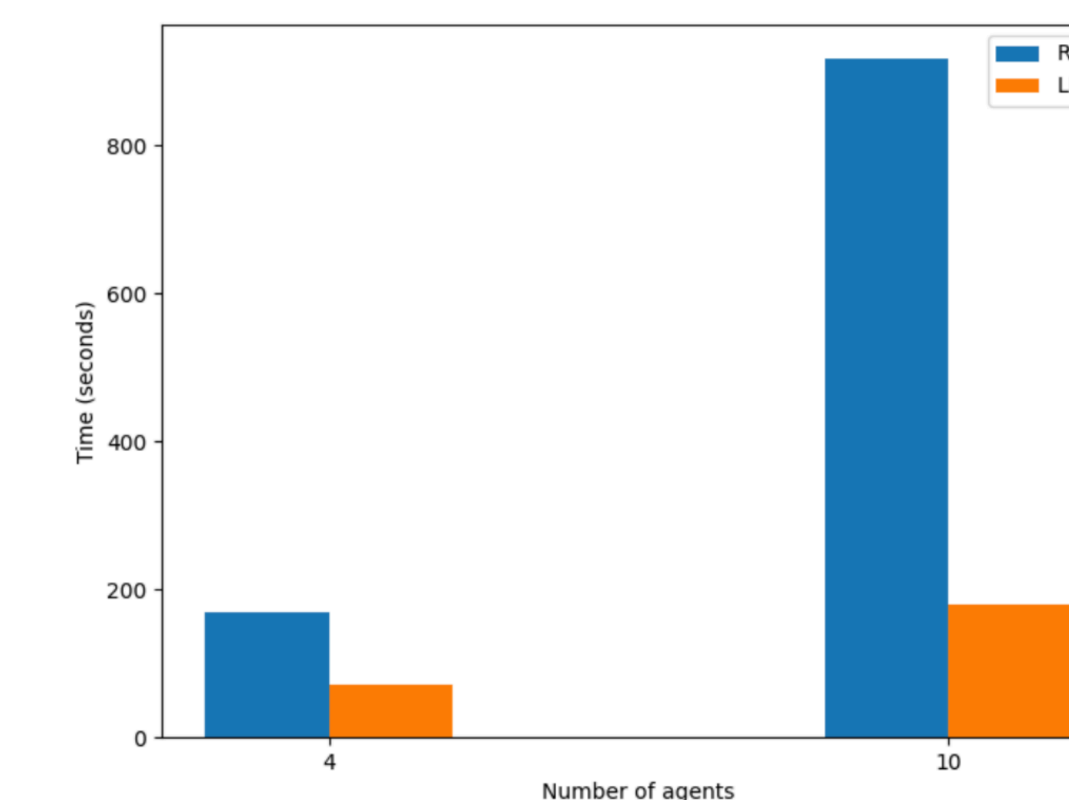
## Experiments



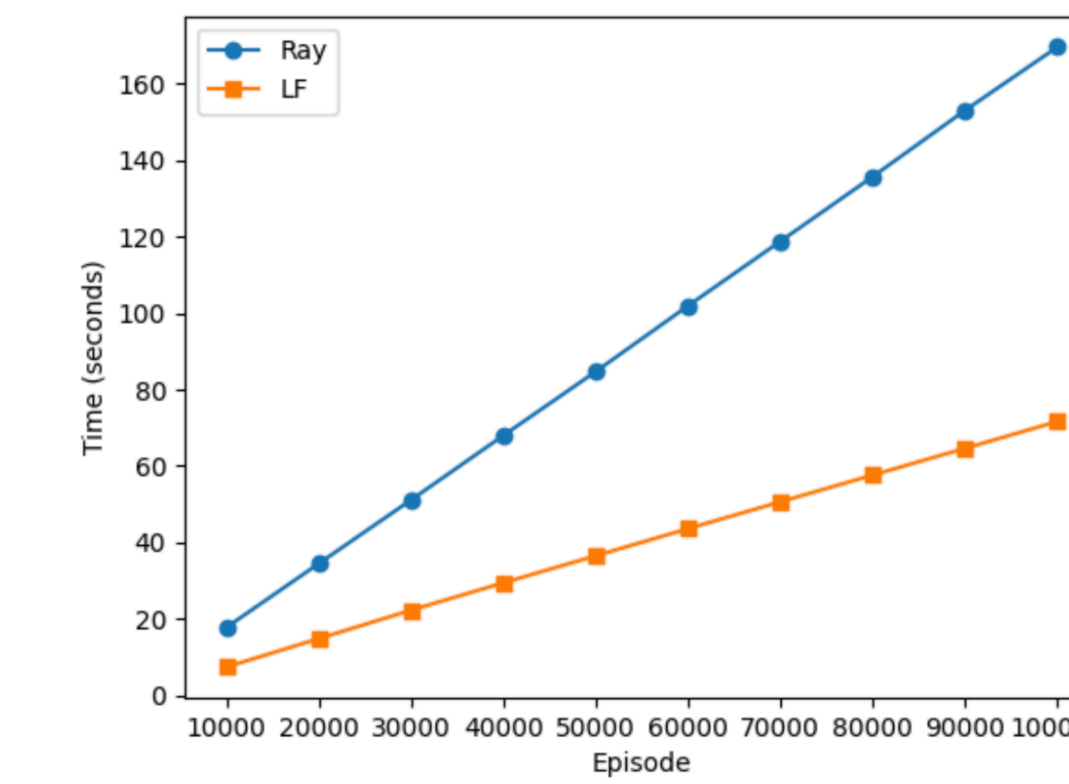
OpenAI Gym Environment Sampling Rates



Atari Environment Sampling Rates



Scaling # agents in multi-agent RL



Communication Overhead vs Episodes

[1] MORITZ, P., NISHIHARA, R., WANG, S., TUMANOV, A., LIAW, R., LIANG, E., ELIBOL, M., YANG, Z., PAUL, W., JORDAN, M. I., ET AL. Ray: A distributed framework for emerging {AI} applications. In 13th USENIX symposium on operating systems design and implementation (OSDI 18) (2018), pp. 561–577.

[2] Menard, Christian, et al. "High-performance deterministic concurrency using lingua franca." ACM Transactions on Architecture and Code Optimization 20.4 (2023): 1-29.