# CSCI599: Life Simulator

Team: ZLX

Zhihan Zhang

Chuanzhe Li

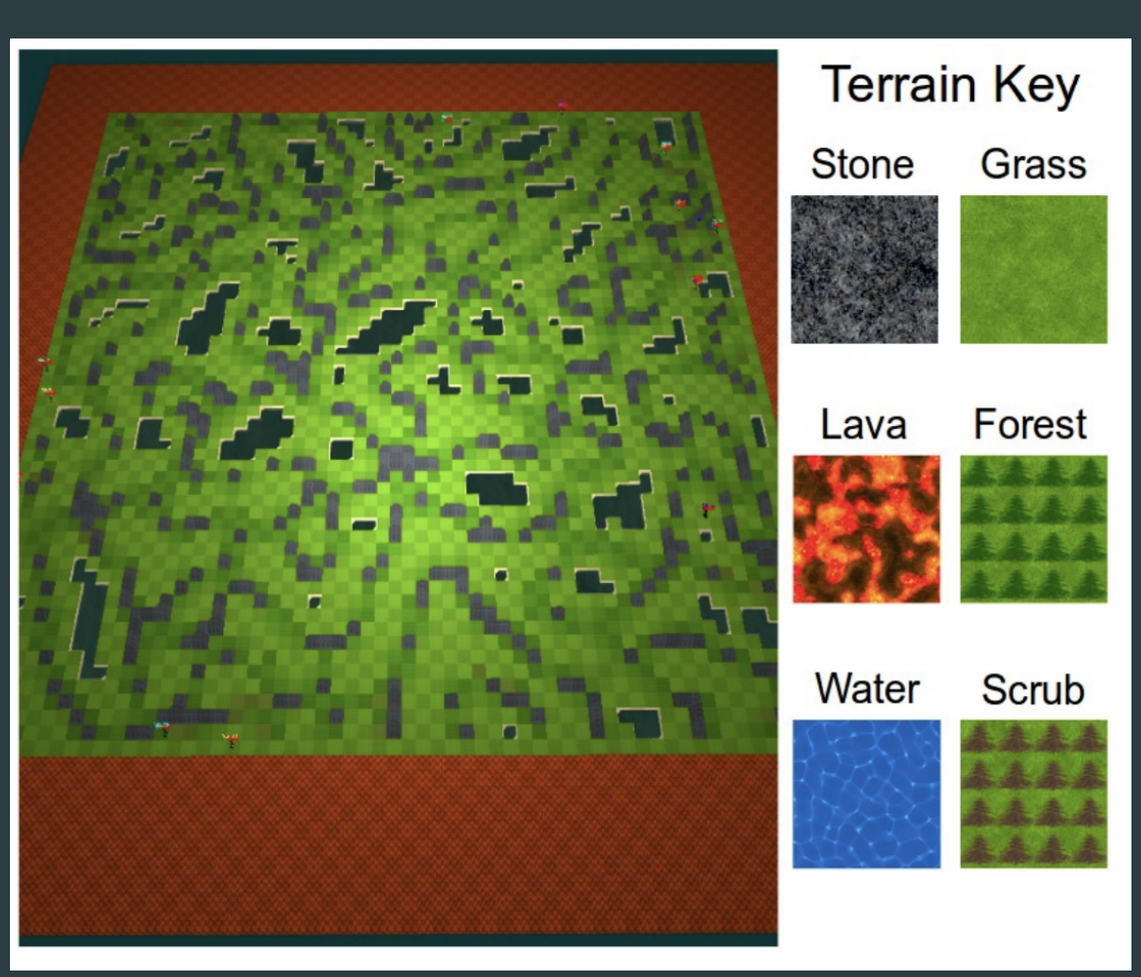
Jia Xu

# Introduction

**Genre:** Neural Massively Multiplayer Online Role-Playing Games (MMO)

**Character:** AI agents

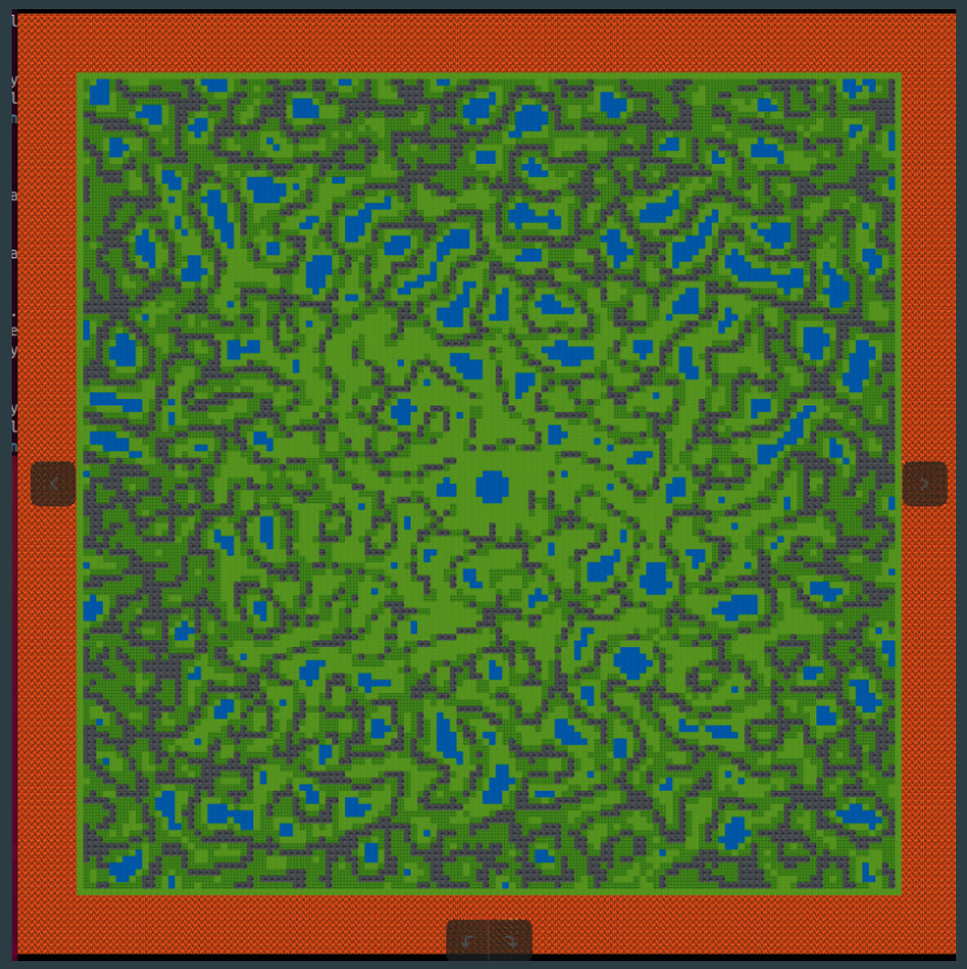
**Purpose:** simulate the evolution of a large number of players competing to maximize survival time in a resource-limited environment



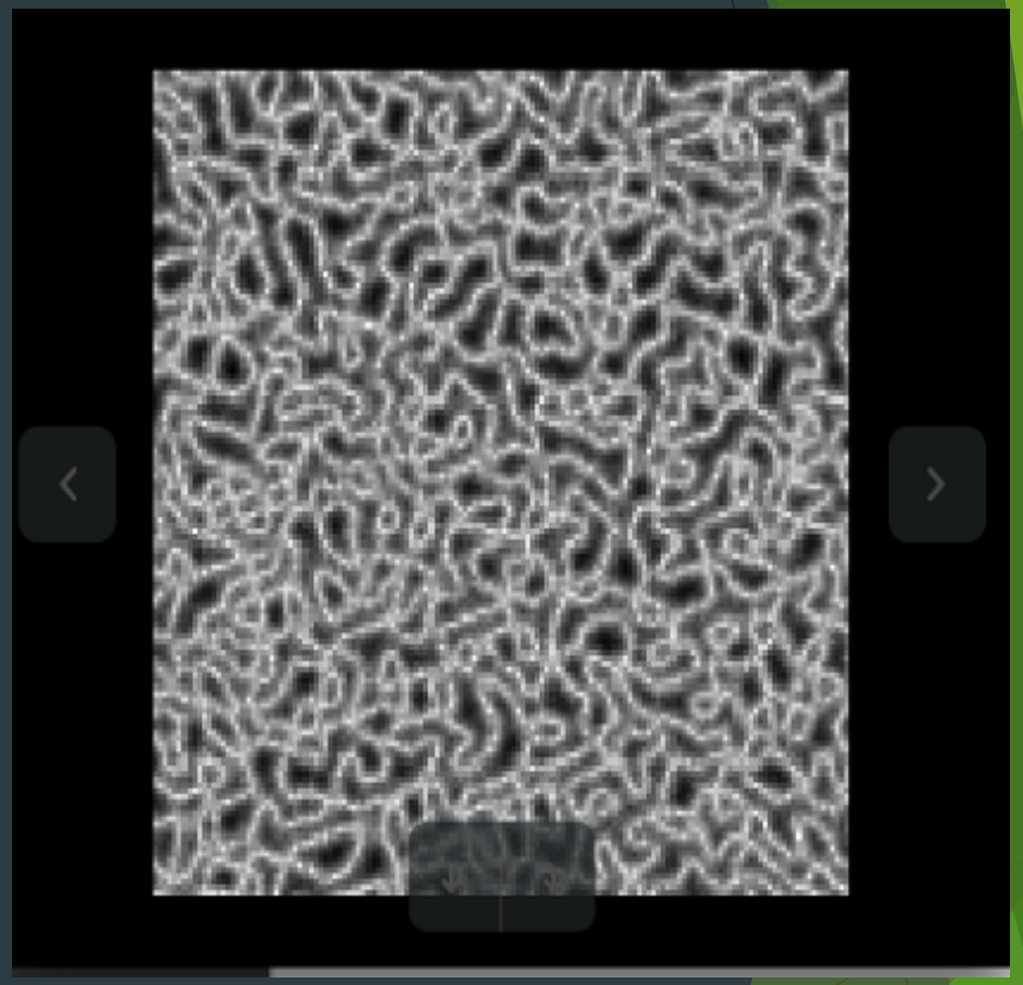

**Map terrain:**

- **Stone**: inaccessible
- **Grass** & **Scrub**: traversable
- **Lava**: die once enter
- **Forest**: traversable, get 5 points food, resource has 2.5% to regenerate
- **Water**: inaccessible, can get water near it

# Introduction



Tile-based terrain map



Texture  map

# Introduction

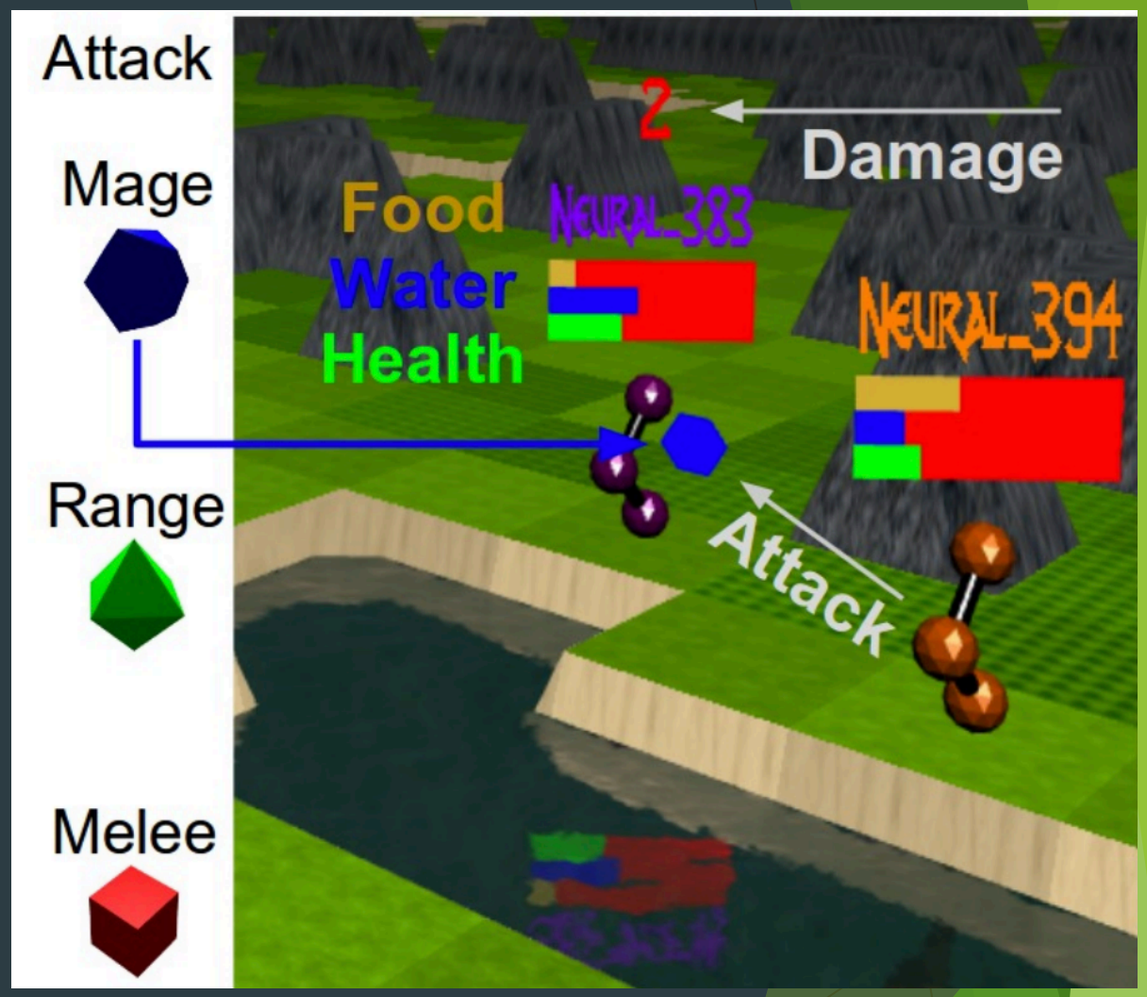
Agent's attributes:

- **Food**: Agents die at 0 health (Yellow)
- **Water**: Agents begin taking damage at 0 food or water (Blue)
- **Health**: Agents begin taking damage at 0 food or water (Green)

# Introduction

## Combat system:

- **Mage**: Inflicts 10 damage at 1 range
- **Range**: Inflicts 2 damage at 1-2 range
- **Melee**: Inflicts 1 damage at 1-3 range and freezes the target in place, preventing movement for two ticks



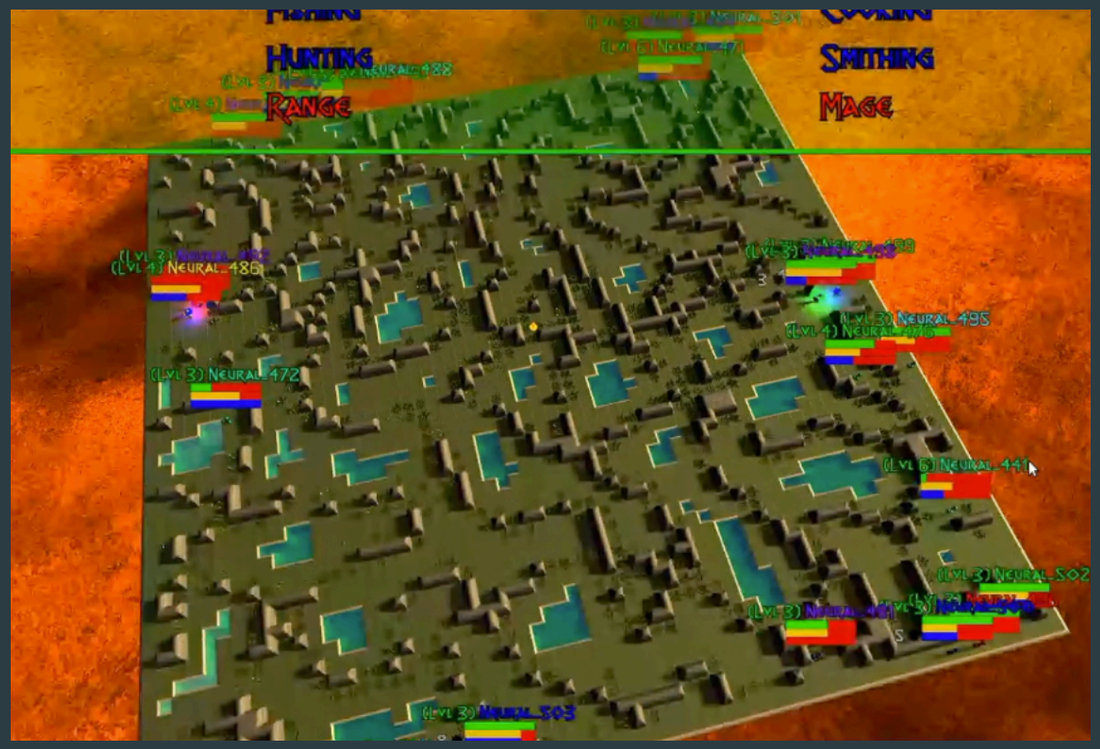
# How we build it

# UI

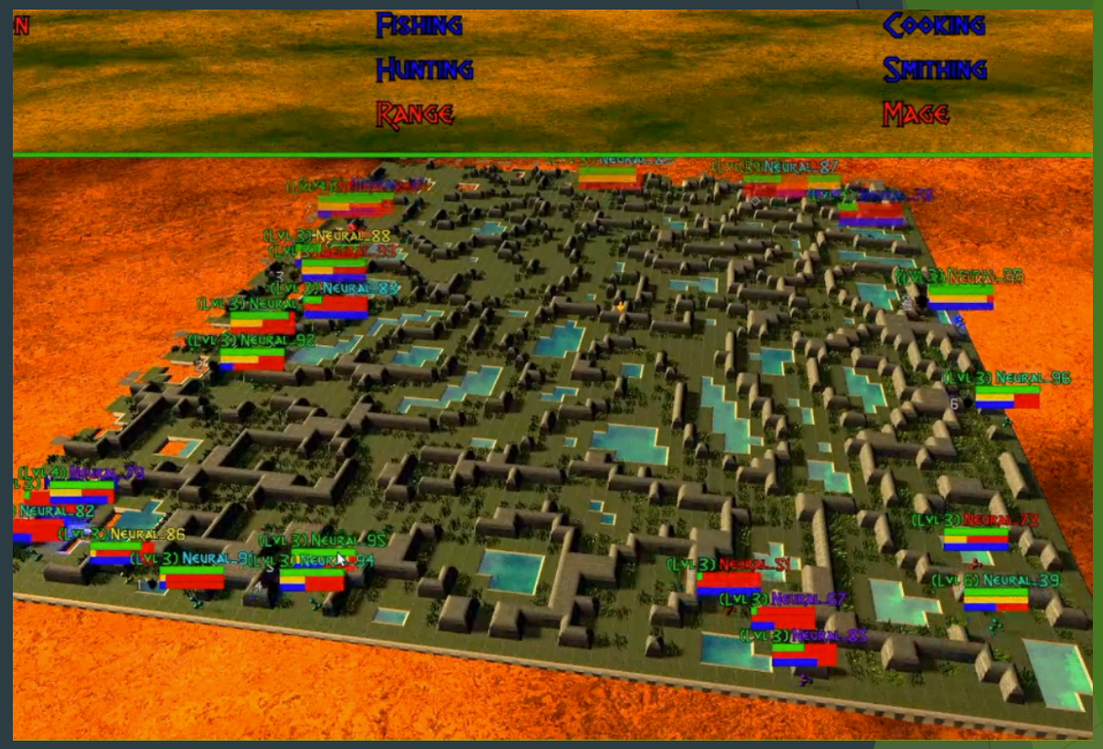
- Unity client UI is built with UI assets implemented by Unity hub
- Optimized the network connection setting

# UI – changeable map size

- Changed UI map drastically to support different size of maps with support to much more clients live in the same map




80 * 80




144 * 144

# Architecture and Training – Define terms in our model

- $s_t$ : the current state of given agents given time tick t, state contains position information, current food, water, health etc.
- $o_t$ : the current observation based on one agent.
- $a_t$ : the action made by given agent which have five types, North, West, East, South, Pass
- $\prod_\theta(a_t|s_t)$: Policy(fully observed)
- $\tau$: the trajectory of given agents which is a sequence of observation, action and reward. $\tau = ( o_t, a_t, r_t, \ldots, o_T, a_T, r_T )$
- $R(\tau)$: reward given trajectory which is the sum of each survival one in every time tick with discount which by default is 0.99 $R(\tau) = \sum_t^T \gamma^t r_t$

# Architecture and Training – Training Steps

The purpose is to maximize the reward in given trajectory:

$$\underbrace{p_\theta\left(\mathbf{s}_1, \mathbf{a}_1, \ldots, s_T, \mathbf{a}_T\right)}_{p_\theta(\tau)} = p\left(\mathbf{s}_1\right) \prod_{t=1}^{T} \pi_\theta\left(\mathbf{a}_t|\mathbf{s}_t\right) p\left(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t\right) \tag{1}$$

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \tag{2}$$

Here we want to optimize the learned parameters maximize the expected survival rewards by adopting policy gradient method:

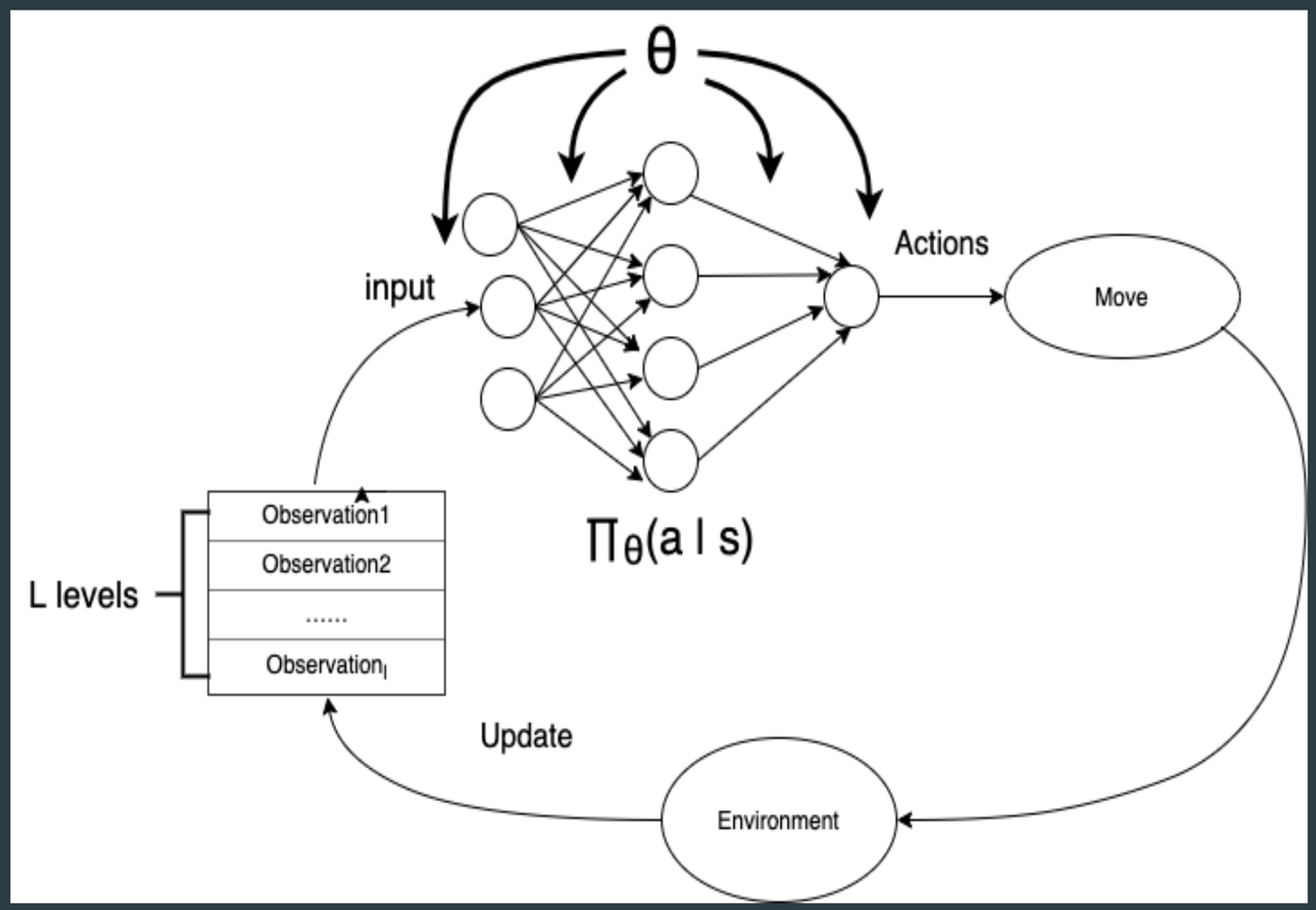
$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \approx \frac{1}{N}\sum_i \sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) \tag{3}$$

$$\nabla_\theta J(\theta) = E_{\tau\sim \pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t|\mathbf{s}_t\right)\right)\left(\sum_{t=1}^{T} r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right)\right] \tag{4}$$

We will update based on epoch:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{5}$$

# Architecture and Training -- Training model



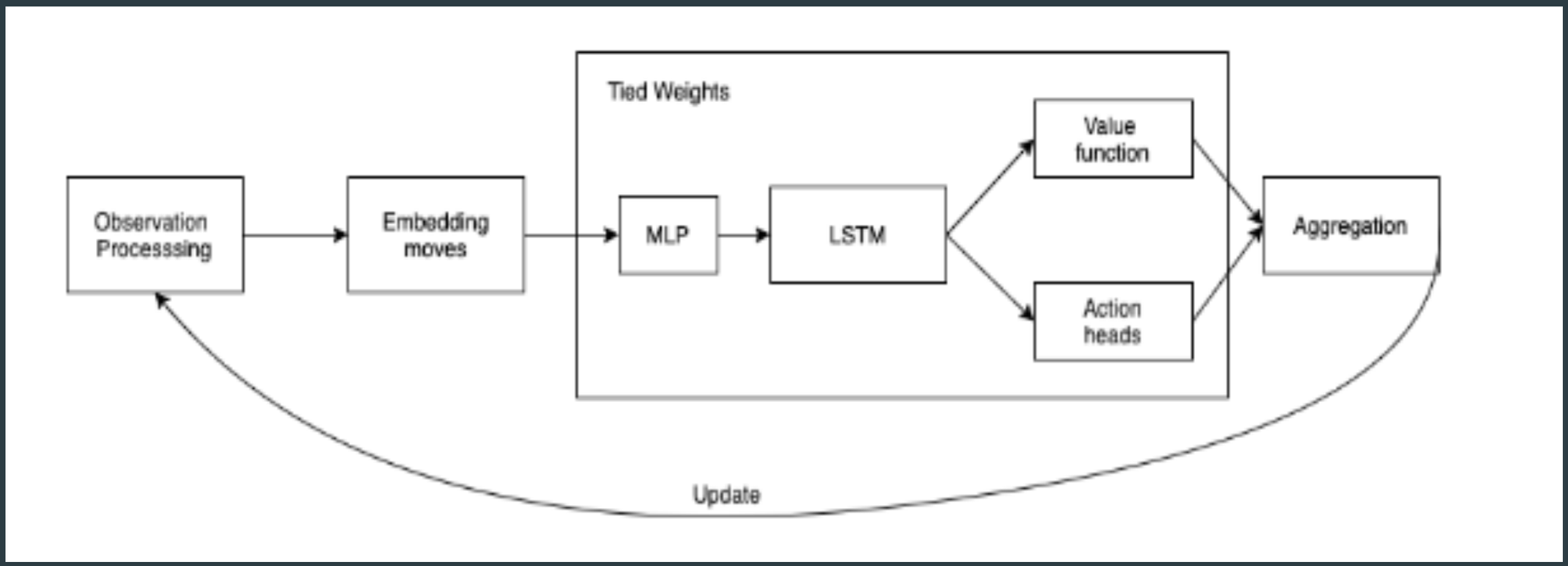

- Linear model
- Use embedding and flatten agents' reward to a one-dimensional encoded vector and fit it into a linear layer of with trainable parameters.

# Architecture and Training -- Training model improvement tried

- Tried many different network layer architectures to optimize the training speed and the performance of agents after training:
  - LSTM layers
  - Attention layers
  - Transformer

# Architecture and Training -- Training model improvement tried

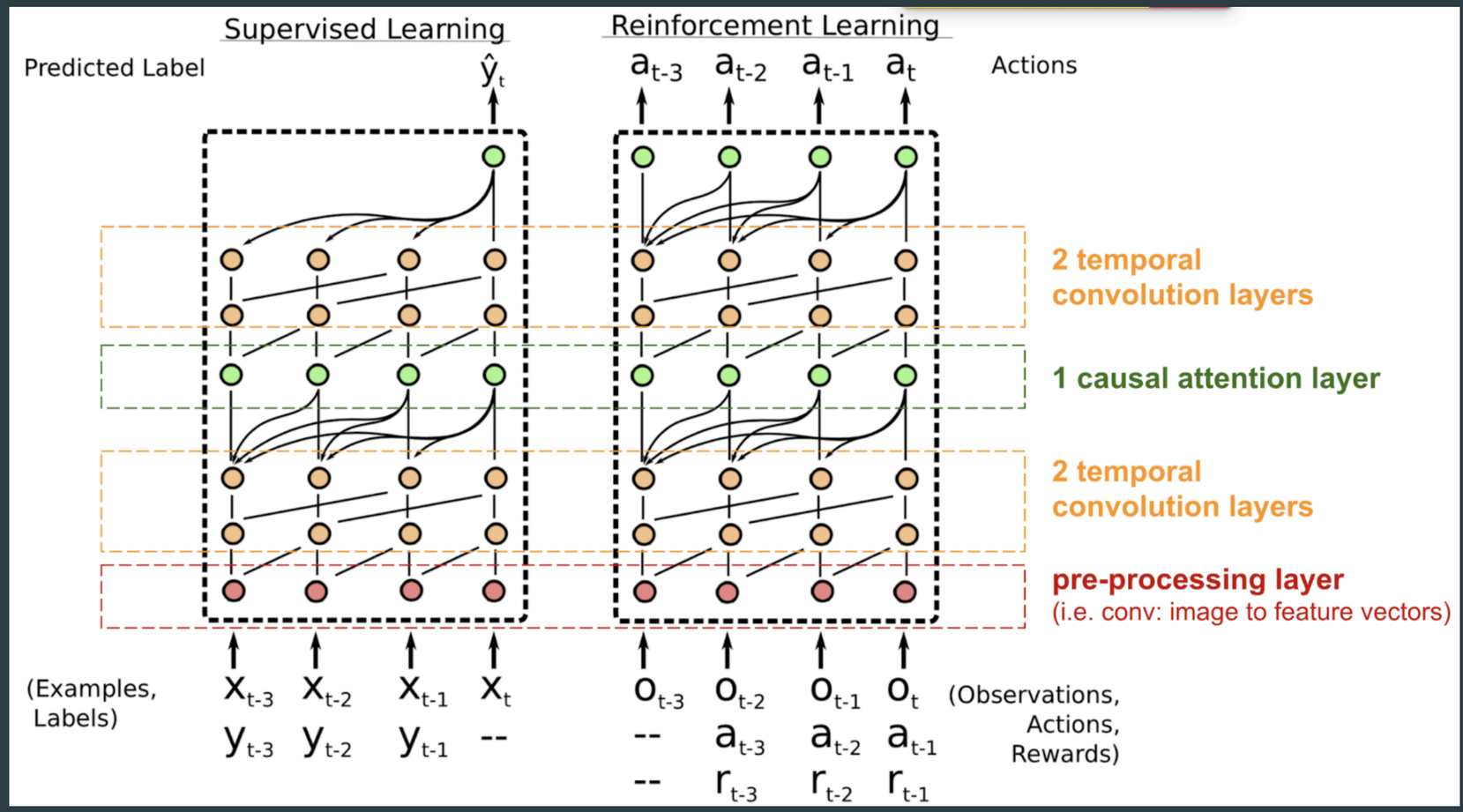
Model with LSTM:




LSTM is widely used in Time series data prediction, thus we want to using it to explore the action in a time-serial based trajectory

# Architecture and Training -- Training model improvement tried

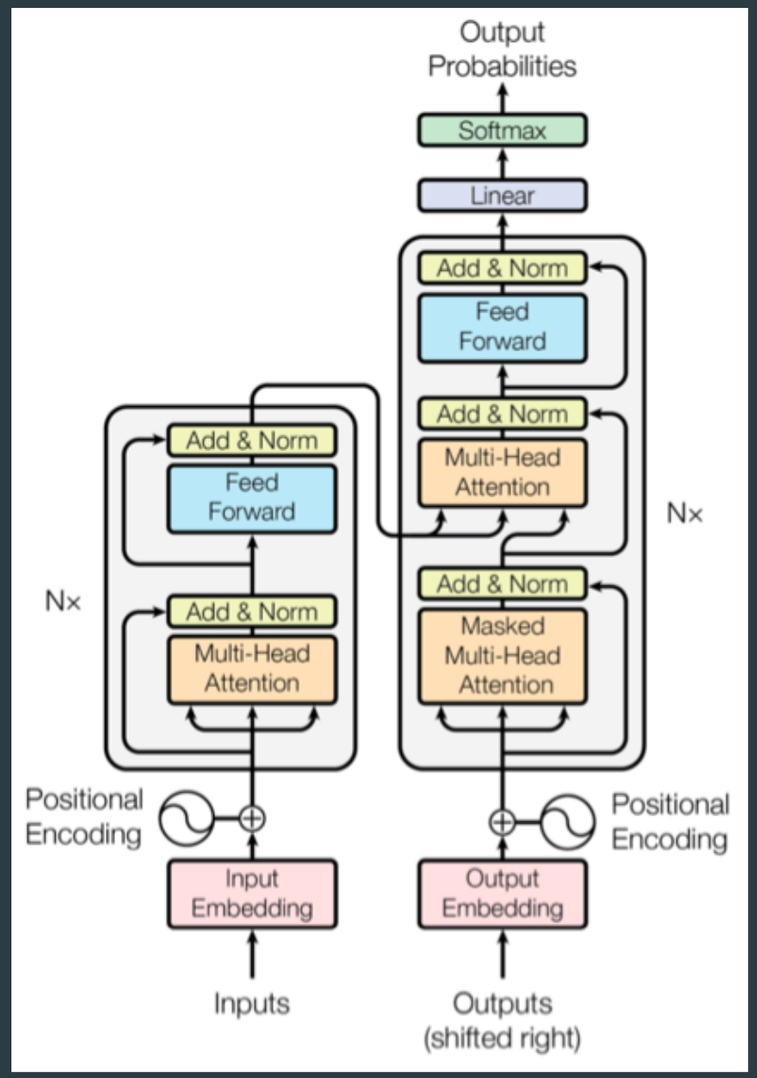
Attention:




Achieve better decision making by adopting global level understanding to the whole trajectory.

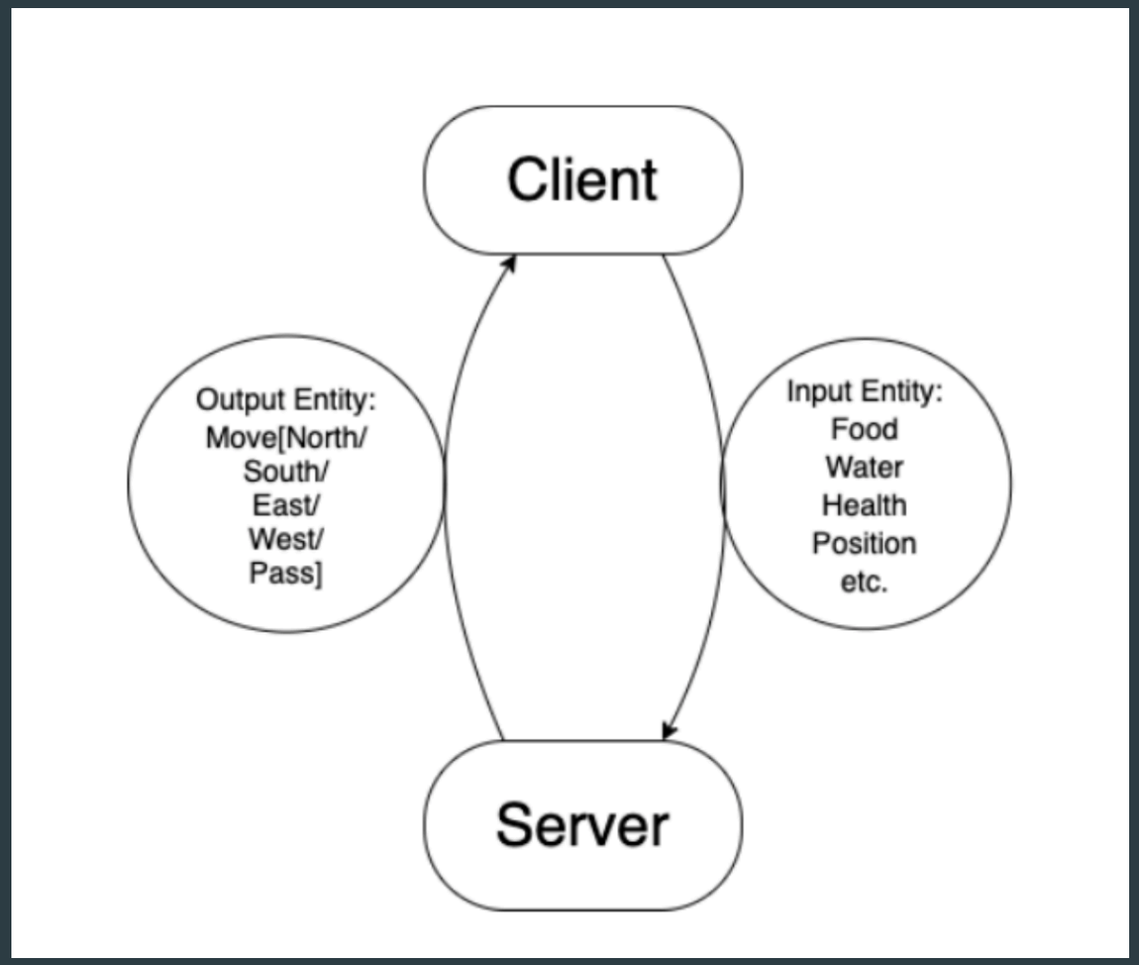
# Architecture and Training -- Training model improvement tried

Transformer:




Neural machine translation may be applicable to our neural network because trajectory information may be viewed as a foreign language sentence.

# Communication network -- Basic network architecture




- Use Twisted Network Library to communicate packet information between client and server

# Communication network -- scaling network  with parameter server

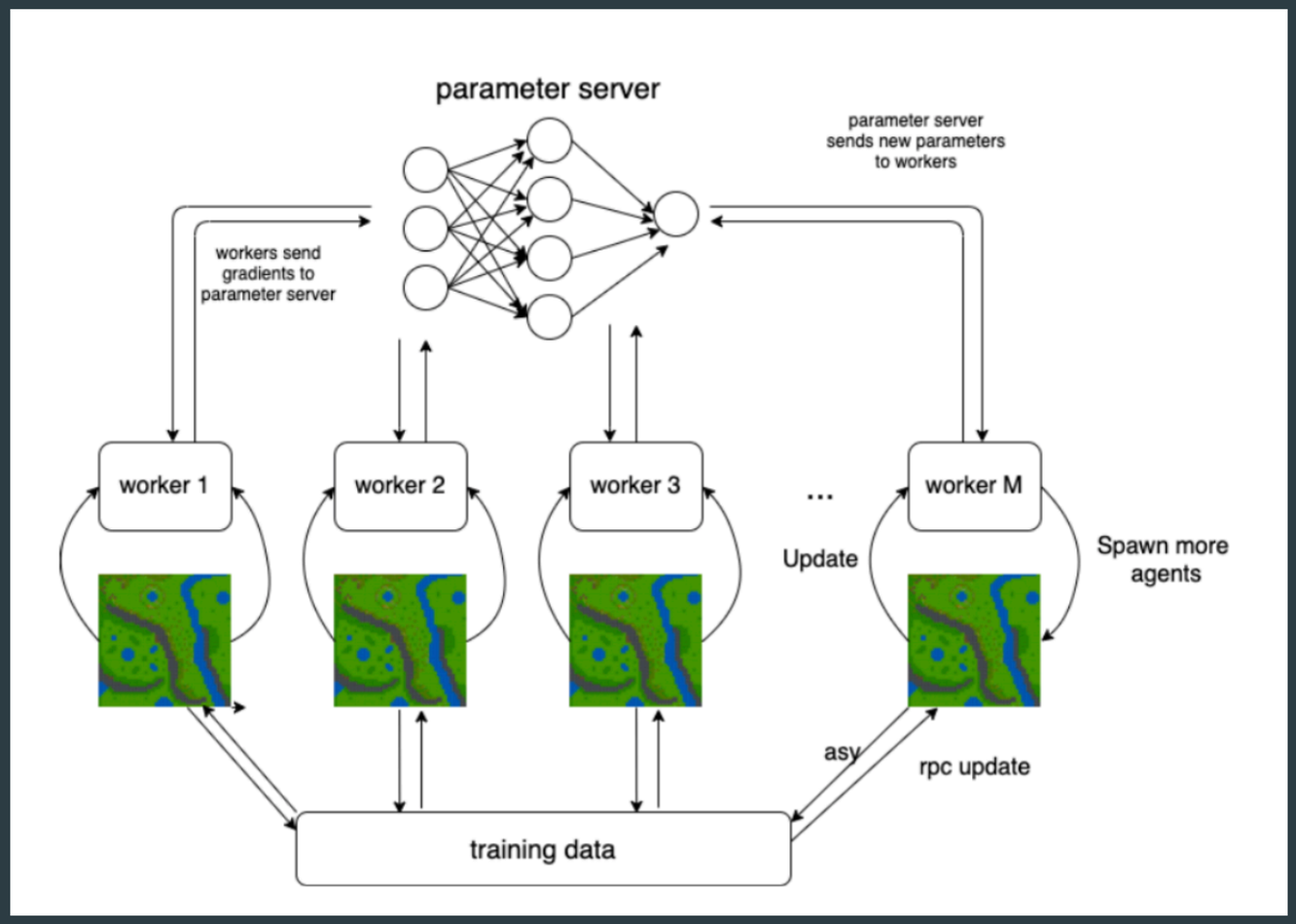
- The initial training model with a single-layer neural network took several days to get results.

- Consider methods to optimize the algorithm by distributing training to different processes and utilizing the computing resources completely.

- The key principle of parallel computing is to overlap computation and communication to avoid high latency.

- Using a parameter server helps us to solve this problem

# Communication network -- scaling network with parameter server



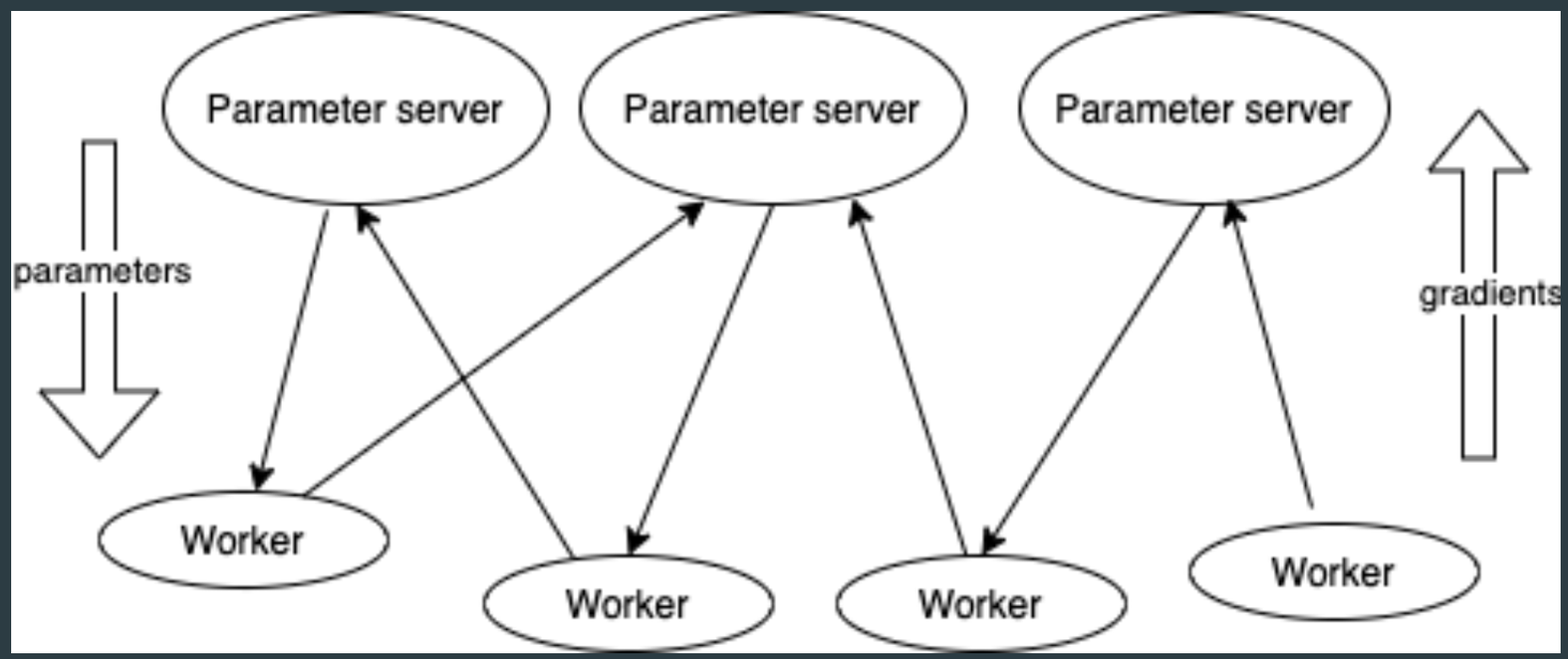

- Step 1: The param server get the parameters computed and pushed by workers
- Step 2 : Param server Broadcasts its updated parameters to all the other worker machines
- Step 3: Workers use the updated parameters to compute gradients.

# Communication network –- Implement parameter server with Ray

- What’s Ray? -- A high-performance distributed framework for large-scale machine learning and reinforcement learning applications.
- Why Ray? –- To simplify the development of high-performance distributed application that runs efficiently on a cluster.

# Communication network -- Implement parameter server with Ray

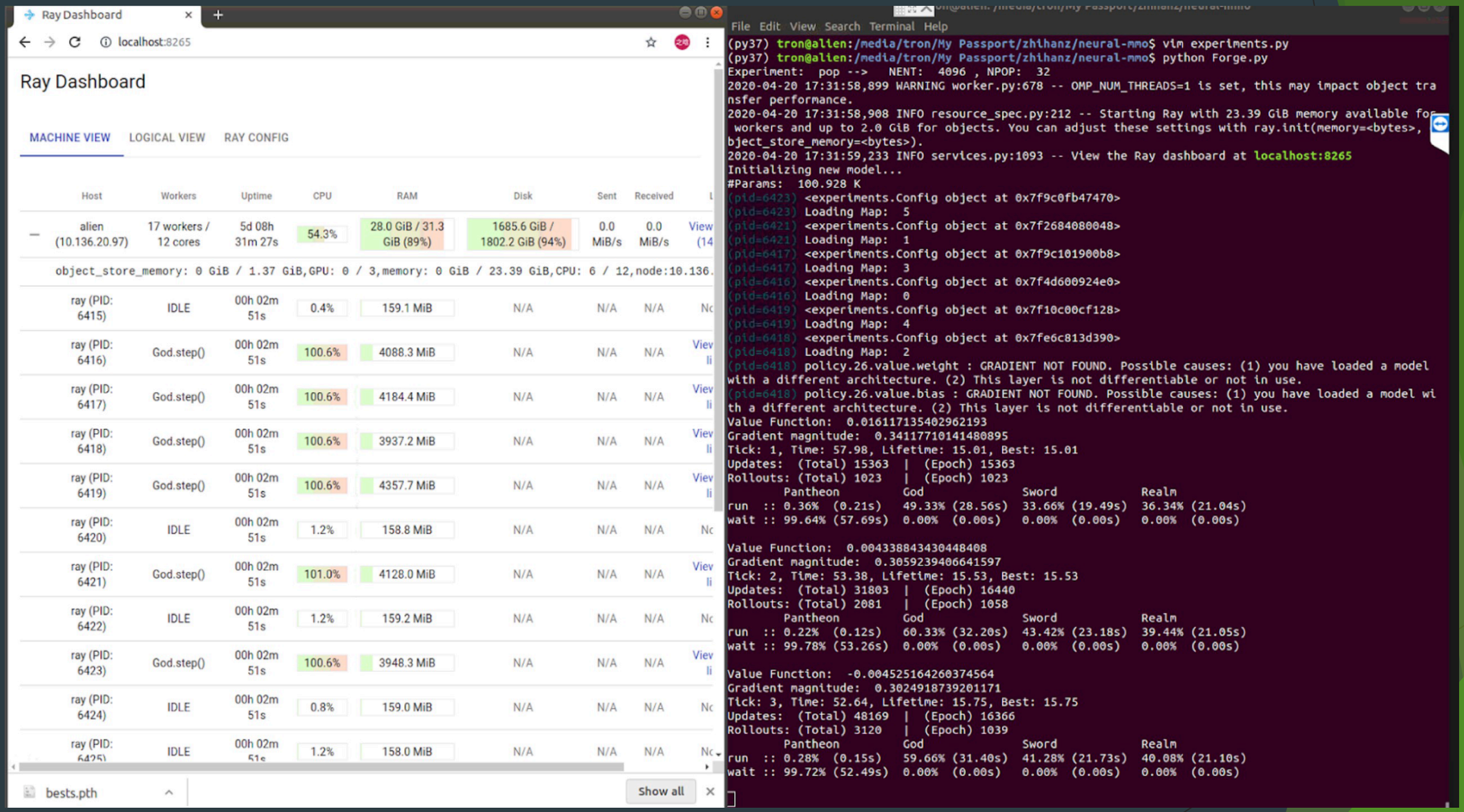
- With the Ray framework, we implement the parameter server pythonically in only a few lines of code, also, it allows us to scale our algorithm that runs on a laptop into a high-performance distributed application that runs efficiently on a cluster with lightweight APIs



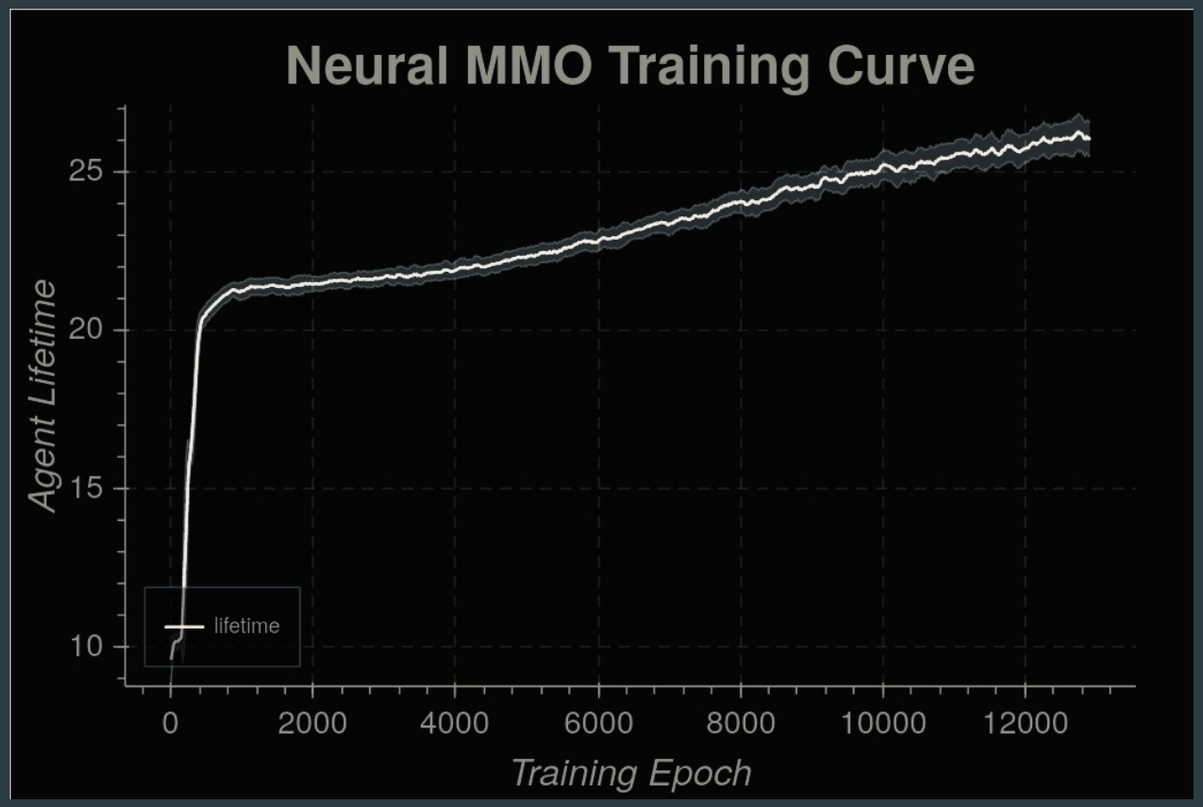
# Communication network -- Implement parameter server with Ray

```
@ray.remote
class ParameterServer(object):
    def __init__(self, keys, values):
        # These values will be mutated, so we must create a local copy.
        values = [value.copy() for value in values]
        self.parameters = dict(zip(keys, values))

    def get(self, keys):
        return [self.parameters[key] for key in keys]

    def update(self, keys, values):
        # This update function adds to the existing values, but the update
        # function can be defined arbitrarily.
        for key, value in zip(keys, values):
            self.parameters[key] += value

@ray.remote
def worker_task(parameter_server):
    while True:
        keys = ['key1', 'key2', 'key3']
        # Get the latest parameters.
        values = ray.get(parameter_server.get.remote(keys))
        # Compute some parameter updates.
        updates = time.sleep(1)


         # Update the parameters.
         parameter_server.update.remote(keys, updates)

 # Start 4 long-running tasks.
 for _ in range(4):
     worker_task.remote(parameter_server)
```

# Communication network -- Implement parameter server with Ray

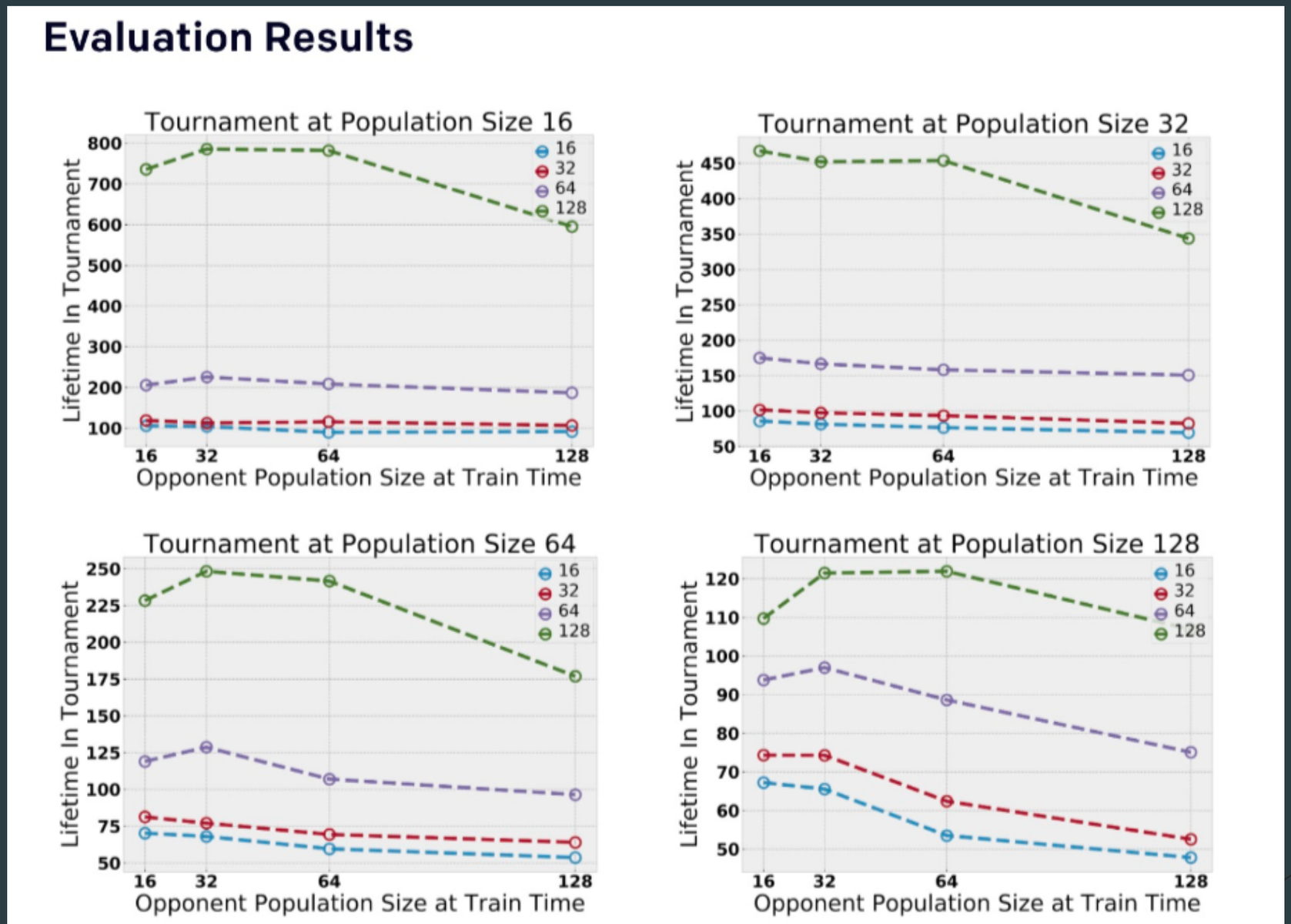


# Training result and analysis




- The baseline model gets to on average > 28 lifetime after training for several days on 12 cores.
- Agents do learn something after training, like not to run into lava first, as indicated by the steep initial learning curve slope.

# Training result



# Limitation

- Map size rendering optimization (currently the model FPS drop drastically if size become 500*500) (show later in the demo)

# Future Work

- Map size rendering optimization
- Add more terrain setups and explore of environment complexity on population behavior (like will people can choose to live near river and generate collaboration instead of competition)
- Deploy AI training to cloud and realize cloud native AI training, deployment and visualization pipeline

# Demo

- https://www.youtube.com/watch?v=5qFh01Kd6vg&feature=youtu.be

# Thank you !