# Deep Reinforcement Learning for Memory-Driven Navigation in Predator-Prey Environments

**Felix B. Berg**
MIT
Cambridge, MA 02139
felixbb@mit.edu

**Filip T. Stromstad**
MIT
Cambridge, MA 02139
filipts@mit.edu

**Ulrik Unneberg**
Harvard University
Cambridge, MA 02138
ulrikunneberg@g.harvard.edu

## Abstract

This project investigates the role of memory in reinforcement learning (RL) by developing a novel framework for memory-driven navigation in dynamic environments. Inspired by natural foraging behaviors, we implement a Long Short-Term Memory (LSTM)-based RL agent trained with Proximal Policy Optimization (PPO) in a 2D predator-prey grid world. The agent relies solely on sensory inputs, without explicit positional awareness, to navigate and survive. Our experiments demonstrate that agents with memory significantly outperform their non-memory counterparts, particularly in complex scenarios where spatial memory aids decision-making. We further analyze the LSTM's hidden states, confirming their encoding of spatial and temporal dependencies critical for adaptive behaviors such as revisiting resources and avoiding predators. These results underscore the potential of memory mechanisms in RL to enhance agent adaptability in dynamic and uncertain environments.

## 1 Introduction

In the natural world, navigation and survival often require organisms to rely on memory to recall the location of resources and threats. Inspired by this, our project explores the use of RL combined with LSTM to mimic such behavior. Unlike traditional RL agents, which often rely on explicit positional data, our approach uses memory mechanisms to guide agents in a 2D grid-world simulation. The motivation behind this is to mimic animals with only sensory inputs, and no knowledge of their absolute position.

### 1.1 Related works

There already exists a significant amount of research within RL for 2D grid worlds [11, 6, 2, 5, 10]. Works like Tizhoosh [12] and SunWoo and Lee [10] have explored path search and navigation in grid worlds using RL, focusing on performance optimization and environment-specific tuning.

The most commonly cited paper for LSTM was published in 1997 [3] and LSTM has since been an area of active research [13, 7, 1]. The introduction of LSTM revolutionized sequence modeling by enabling networks to capture temporal dependencies. This concept has been widely adopted in RL, particularly in tasks requiring memory or temporal context. More recent works, such as those by Grzelczak and Duch [2], integrate deep RL with LSTM architectures for path planning and memory modeling.

Works like Kyaw et al. [4] delve into dynamic path planning but lack emphasis on memory mechanisms that mimic natural learning behaviors. To the best of our knowledge, there has been little research on the use of LSTM to model memory in an agent that needs to survive in a 2D environment. Our research seeks to address this gap by combining RL with LSTM in predator-prey environments, enabling agents to encode spatial memory without explicit positional information.

### 1.2 Contributions

The choice of the problem formulation and RL aspects under investigation were motivated by exploring how animals learn spatial navigation through evolution. We achieve this by denying the use of explicit positional data and instead focus on sensory inputs and memory mechanisms in LSTM for spatial learning. In addition to looking at LSTM policies and feed-forward policies, we also investigate a combination of the two to see if a good combination of instinct and memory can be achieved. This is the novel part of our work.

With this novelty, we were able to perform analysis of the hidden states of the LSTM layer to see if it encodes memories of past positions and impressions, or plans about future movement. Since we have created our own environment, we have been able to test different combinations of environments, policy architectures, learning methods, reward functions and exploration features. We hope to contribute with significant analysis to understand how these combinations affect performance, and how using memory compares to regular instinctive feed-forward policies in different scenarios.

## 2 Problem formulation

In the following, we describe how we model our problem, and how we solve it. The first component in any RL problem is the environment - how the world in which the agent lives behaves. Then, we describe how we model states, actions and rewards. Lastly, we describe how we train the agent.

### 2.1 Environment, States & Actions

We have created a 2D grid world simulation environment, illustrated in Figure 1. This environment consists of $n \times n$ tiles with boundaries preventing movement outside the domain. The agent interacts with the environment where it can consume apples. Apples can appear either as standalone entities or within predefined apple trees, depending on the desired environmental configuration. The set of apple trees can be placed either at fixed positions or randomly generated locations within the environment. A fixed number of predators move to neighboring tiles, with their direction chosen uniformly when far from the agent, but heavily biased toward the agent when nearby. Both the predator and agent can move in four directions. The agent lacks explicit positional knowledge, simulating a natural learning process. To survive, the agent must find food (apples) to avoid starvation, and must not be captured by predators. The agent's goal is to survive for as long as possible.

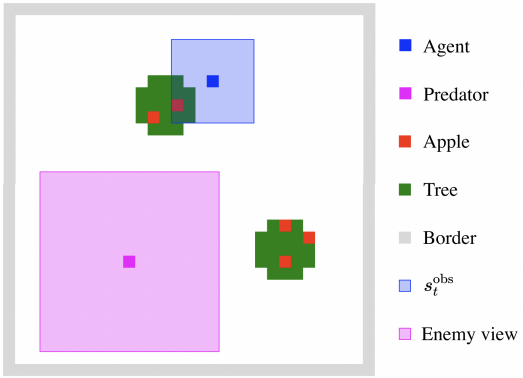



Figure 1: Design of environment, where the agent and the predator has limited sight.

At each time step $t$, the environment is in state $s_t = (s_t^{\text{obs}}, s_t^{\text{unobs}}) \in \mathcal{S}$, the agent takes an action $a_t \in \mathcal{A}$, and receives a reward $r_t$. The partially observed state, $s_t^{\text{obs}}$, includes $n_{obs}^2$ neighboring tiles, which can be {*apple*, *apple tree*, *predator*, *boundary*, *nothing*} and its hunger $H_t$. The remaining tiles make up $s_t^{\text{unobs}}$. Actions include moving {*up*, *down*, *left*, *right*}, and transitions are deterministic (e.g., "up" moves the agent to the tile above). Hunger increases linearly with time since the last meal. Eating an apple provides reward $r^{\text{apple}}$ and resets $H_t = 0$. Encountering a predator results in a terminal state $s_{\text{term}}$ with a large negative reward, simulating death. The reward at time $t$ is then given by

$$r_t = \begin{cases} r_{\text{min}} + (r_{\text{max}} - r_{\text{min}}) \cdot \frac{H_t}{H_{\text{max}}} & \text{, if eating apple at } t \\ -r_{\text{max}} & \text{, if eaten or die of hunger at } t \\ 0 & \text{, else.} \end{cases}$$

We tested multiple reward functions but ended up with this as the baseline. This reward function managed to induce interesting behavior, and it relies on senses rather than e.g giving the agent an explicit fear of being near the predator.

### 2.2 Policy Architecture

The policy is modeled as shown in Figure 2. The input layer is the observable states, $s_t^{\text{obs}}$. Via one hidden layer, the information is fed into an LSTM layer. We then use an actor-critic network with two hidden feed-forward layers. The action is determined by a softmax over the actor-head. The critic network is similar but only consists of one output node $\hat{V}(s_t^{\text{obs}})$. The number of neurons in the layers, and the number of layers are tuning parameters that we optimized during testing.

The reason that we include an LSTM layer is to introduce memory to the agent. LSTM neurons are specialized types of neurons capable of maintaining hidden states over time. Unlike standard feed-forward neurons, which process information in isolation, LSTM neurons can capture temporal dependencies by selectively retaining or discarding information through gates: the input gate, forget gate, and output gate. This ability to manage memory allows the agent to utilize information from previous observations or actions to make more informed decisions. In our environment, this could be remembering the location of apple trees.

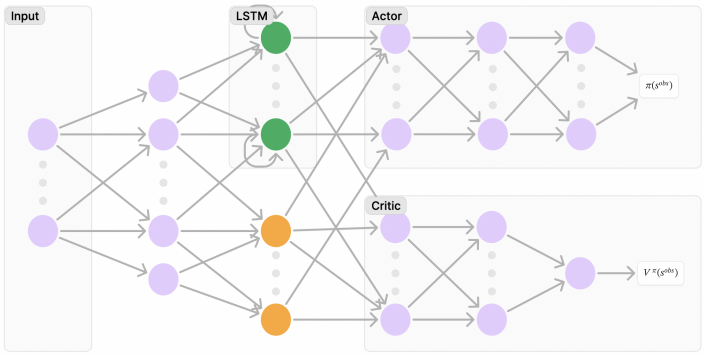



Figure 2: Neural Network policy architecture

As we will discuss later, the number of LSTM neurons (green) is one of the interesting parameters to vary. The sum of the green and the orange neurons is constant among all experiments. When training without memory, we replace the LSTM layer with a regular feed-forward layer.

### 2.3 Training strategy

We decided to use the clipped Proximal Policy Optimization (PPO) with $\epsilon = 0.2$ to train the agent, as it has better overall performance than other state of the art models [9]. This also avoids the use of extensive hyperparameter tuning. The clipped loss function is given by

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right], \quad r_t(\theta) = \frac{\pi_\theta(a_t | s_t^{obs}, m_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t^{obs}, m_t)}.$$

We performed parallel training by simulating $m$ agents simultaneously to reduce variance in policy updates and improve sample efficiency. To estimate the advantage, we use the General Advantage Estimator (GAE) as described in [8] for variance reduction and its proven good results. To encourage exploration, we include entropy regularization.

### 2.4 Implementation

In general, we have built the entire reinforcement learning setup from scratch, but used `PyTorch` as the machine learning and optimization framework. The `PPOAgent` class is the central component responsible for managing the training process. It handles the collection of rollouts from multiple parallel environments, computes advantages using GAE, and updates the policy network using the PPO algorithm. The class maintains LSTM hidden states across episodes, ensuring that temporal dependencies are preserved.

The `PolicyValueNetwork` class defines the neural network architecture used by the agent. It consists of an LSTM layer sandwiched between fully connected layers, with separate heads for policy and value estimation. This architecture allows the network to process sequential data and output both action probabilities and state value estimates.

The environment is managed through the class `GridWorldEnv` that simulates a grid-world scenario. This class handles the dynamics of the environment, including agent movement, resource management, and interactions with other entities like predators and trees.

The `EnvFnWrapper` and worker function facilitate the creation and management of parallel environments, enabling efficient data collection for training. This setup allows the agent to interact with multiple instances of the environment simultaneously, speeding up the training process. The codebase also includes utility functions for visualization, such as plotting reward curves and agent trajectories. These tools are essential for monitoring training progress and analyzing agent behavior.

## 3 Experiments

In the experiment part of this paper, we chose two environments to systematically demonstrate the agents' performance. The first environment is a simple search for an apple, while the other environment is a more complex foraging task. The parameters we used in both experiments are given in Table 1. They were all selected after extensive testing, or found in the aforementioned literature.

| **Parameter** | **Value** | **Description** |
|---|---|---|
| num_envs | 100 | Number of parallel environments used for training |
| num_steps | 256 | Number of steps per environment per update |
| num_updates | 2000 | Total number of policy updates during training |
| hidden_size | 256 | Size of all hidden layers, including in the LSTM |
| grid_size | Ex1= 20, Ex2= 100 | Dimensions of the square grid world environment |
| view_size | Ex=5, Ex2=7 | Total sidelength of the Agent's view square |
| max_hunger | 100 | Maximum hunger value before agent dies |
| num_trees | Ex1=0, Ex2=1 | Number of apple trees in environment |
| num_predators | Ex1= 0, Ex2=1 | Number of predators in environment |
| $\gamma$ | 0.99 | Discount factor for future rewards |
| $\lambda_{GAE}$ | 0.95 | Lambda parameter for GAE calculation |
| $\epsilon_{clip}$ | 0.2 | PPO clipping parameter |
| $\beta_{entropy}$ | 0.01 | Entropy coefficient for exploration |
| $c_1$ | 0.5 | Value loss coefficient in total loss |
| max_grad_norm | 0.5 | Maximum gradient norm for clipping |
| $\alpha$ | $2.5 \times 10^{-4}$ | Learning rate for Adam optimizer |
| $\epsilon$ | $1 \times 10^{-5}$ | Epsilon parameter for Adam optimizer |

Table 1: Hyperparameters and Environment Parameters

### 3.1 Experiment 1 - Small arena, random apples, no predators

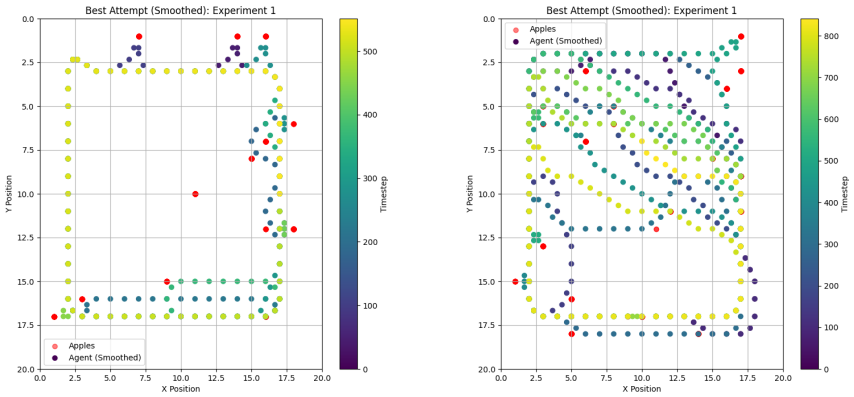
Consider the 20x20 environment with an apple spawning at a random position in the grid world. When the agent finds the apple, it will spawn a new apple at a random position on the map. For this example, we use a simplified reward function with +1 reward for eating an apple and -1 for dying.

As we can see in Figure 3, we see significantly different behaviors when using LSTM versus not. In the first plot, we see that the non-LSTM agent keeps following the walls. It is able to get the apples

along the wall, but once an apple spawns in the middle of the map, it doesn't explore enough to find it and dies. With LSTM, however, it searches the space more thoroughly. This signals that it has some notion about where it has been, and where it should explore to find new apples, making it more efficient.

Specifically, the agent with the LSTM manages to get an average reward of 2.93 after training. This means that the agent on average, manages to eat little under four apples before dying. The non-LSTM agent manages to get a reward of 1.75, which is lower, indicating that it is not as good to search for apples.

Overall, the non-LSTM agent has reasonable behavior. It avoids fumbling in blindness by always walking along the wall, and hopes that it will see apples in its vicinity. Its problem is finding apples that spawn in the center. The LSTM agent on the other hand, uses the walls to gain knowledge about its position and then traverses into open field to look for apples. On average, this covers a greater area, but as we see from the plot (b), it misses an apple in the corner.




(a) Agent trajectory without LSTM (b) Agent trajectory with LSTM

Figure 3: Trace plot from best run of trained models in experiment 1.

### 3.2 Experiment 2 - Large arena, apple tree, with predator

In this experiment, the grid is made so large that it can be approximated as infinite, as the agent never hits the walls in the simulation. The agent spawns randomly on the grid but close enough so that it can see the tree in the middle. Furthermore, the enemies are biased to move towards the middle.

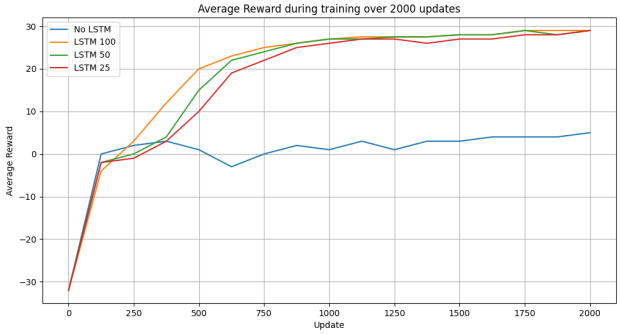



Figure 4: Average rewards during training with 2000 steps

In this experiment, we trained agents with varying number of LSTM neurons in the hidden layer: 0%, 25%, 50% and 100%. As we see from Figure 4, the agents relatively quickly learned beneficial

pattern shown by an average reward of 0 within the first 200 updates. The non-LSTM agent never increased its reward further, and we will study its behavior in more detail later. All the memory-agents managed to learn more complex policies utilizing their memory of the position of the tree and the predator even though it was not in the field of sight. We notice that the agents with the larger LSTM layers were able to learn good policies more quickly. We also notice that the rewards reached some sort of limit, as the apples spawn at a limited rate.

For the trained policies with 0%, 50% and 100% LSTM neurons, we ran 1000 independent simulations to compare their performance. The results are summarized in Figure 5. We can see that the policy with 100% LSTM neurons performed slightly better than the policy using only 50%, but both drastically outperforming the policy without any memory at all. We further examine the performance difference between the policies with and without LSTM by analyzing the simulations with the longest survival times across 1000 runs for both approaches.

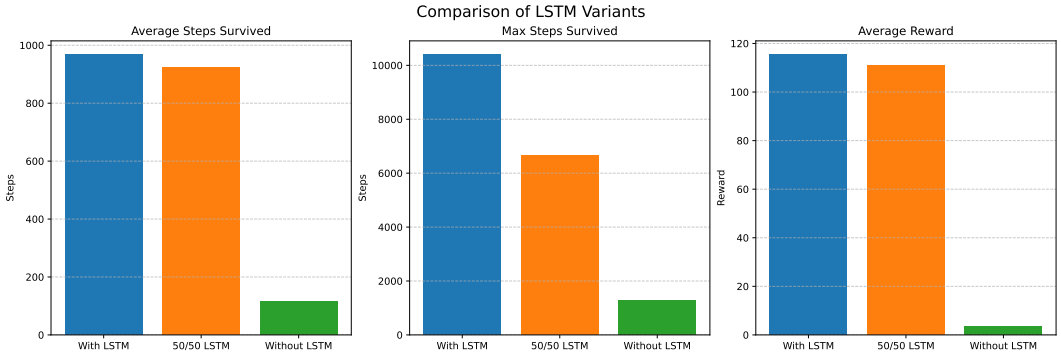



Figure 5: Barplot of the performance from 1000 independent simulations of the agents with LSTM, with 50/50 LSTM and feed-forward, and with no LSTM.

In Figure 6 we can see the agent that survived the longest without having any memory. The agent always makes sure that it can see parts of the tree, as seen by "time since last tree seen" always being zero. This aligns with our expectations. Without any memory, the agent has no way to decide which direction to go for food if it cannot see the tree. This limits the agents available movement space, and it is eaten as soon as the predator enters the tree.

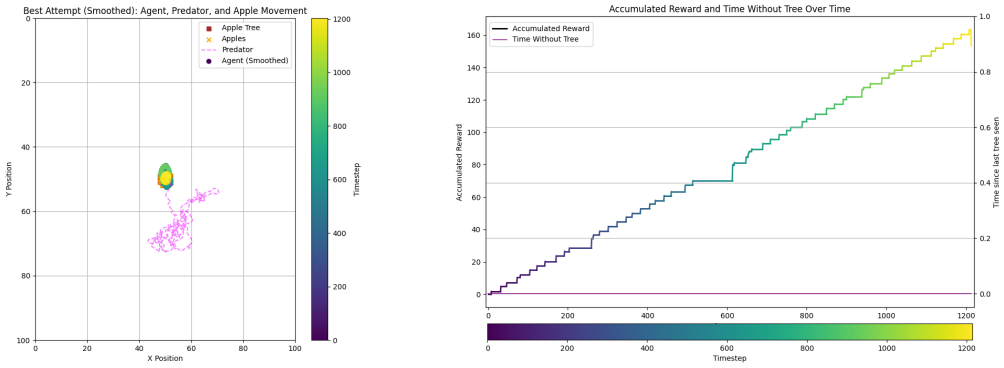



Figure 6: Simulation with most steps survived out of 1000 - without LSTM. **Left:** Trace of agent and predator movement. **Right:** Accumulated reward and time since last tree seen.

In Figure 7 we see the behavior of the LSTM agent that survived the longest. In the left subplot we see the trace of its movement. Compared to the trace of the agent without any memory, we clearly see that the agent is able to move away from the tree. This is showed more explicitly in the right subplot, where the purple lines show the time since the agent last could see any part of the tree. Most of its

lifetime, there is upwards of 20 time steps since it had the tree within view - and it is still able to find back. We see the accumulated reward steadily increase, until the agent is finally eaten by the predator.

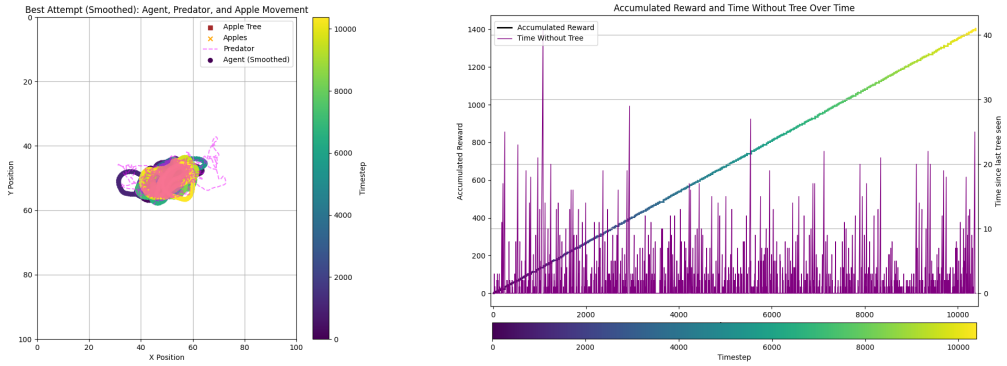



Figure 7: Simulation with most steps survived out of 1000 - with LSTM. **Left:** Trace of agent and predator movement. **Right:** Accumulated reward and time since last tree seen.

In Figure 8 we zoom in on a specific part of the trajectory shown in Figure 7. This is the part between timesteps $1000$ and $1100$ which we can recognize as the clearly visible loop in the left subplot in Figure 7 and as the tallest spike in the right subplot of Figure 7. Here, we see the agent being chased away from the tree by the predator. The agent then makes a loop, and heads straight back to the tree - which at this point is far out of its observable state, and has been for over $40$ time steps. This is just one example of many from this run, where the agent is able to get back to the tree, without actually seeing the tree. This indicates its ability to use short term memory.

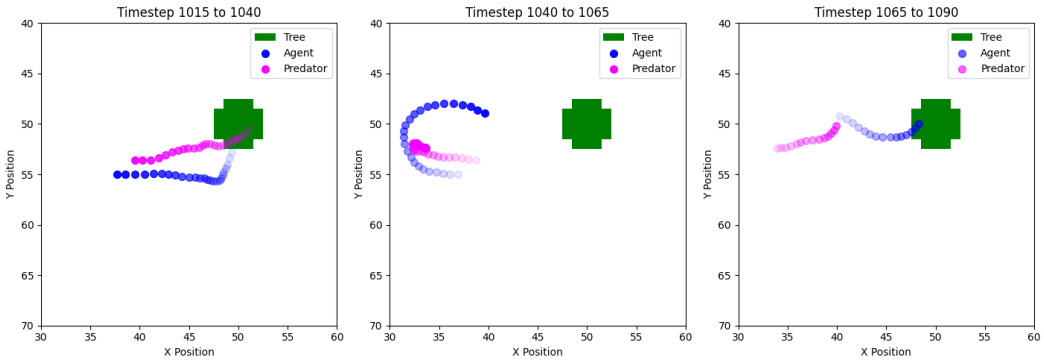



Figure 8: Trajectory of LSTM agent when navigating back to the tree

### 3.3 Memory

We investigate the agent's memory and ability to predict future, current and past positions. If an agent can encode this information, the agent has the possibility to use position, and past expression to generate a plan for future movement.

In order to decode the memory, we load the policies from the 25% and 100% LSTM agents from experiment 2, and generate 100 episodes for each policy to collect all hidden states and positions of the agent in a csv with length around 100,000. We split this into training data and test data. The decoding strategy involves trying to predict the position $Y_{t+\Delta t}$ of the agent $\Delta t$ timesteps in the future or past from time $t$. The space of hidden states $h_t$ represent the feature space. We want to find the

function $f$ that maps $\vec{h}_t$ to $Y_{t+\Delta t}$:

$$Y_{t+\Delta t} = \begin{bmatrix} x_{t+\Delta t} \\ y_{t+\Delta t} \end{bmatrix} = f(\vec{h}_t), \quad h_t \in \left\{\mathbb{R}^{64}, \mathbb{R}^{256}\right\},$$

where the size of 64 or 256 depends on whether we evaluate the 25% LSTM agent or the 100% LSTM agent.

Certain neurons have very little activation and the coefficients of $f$ can get very large making $f$ ill-conditioned. We therefore include a penalty term for large coefficients in this loss function. For computational speed we chose ridge regression. The form of the loss function is therefore

$$L(f) = \sum_{i=1}^{N} (Y^i_{t+\Delta t} - f(h^i_t))^2 + \alpha \|f\|^2_K, \quad f(h_t) = Ah_t + b, \quad A \in \mathbb{R}^{2\times\{64,256\}}, \quad b \in \mathbb{R}^2.$$

Using this function on the test set of $h_t$'s we can predict positions and compare to the agent's true position. We did the memory analysis for experiment 2 because it was interesting that the agent was able to find its way back to the tree. We hypothesized that it would have an internal representation of the tree's position in relation to itself. The results were promising, as can be seen in Figure 9.

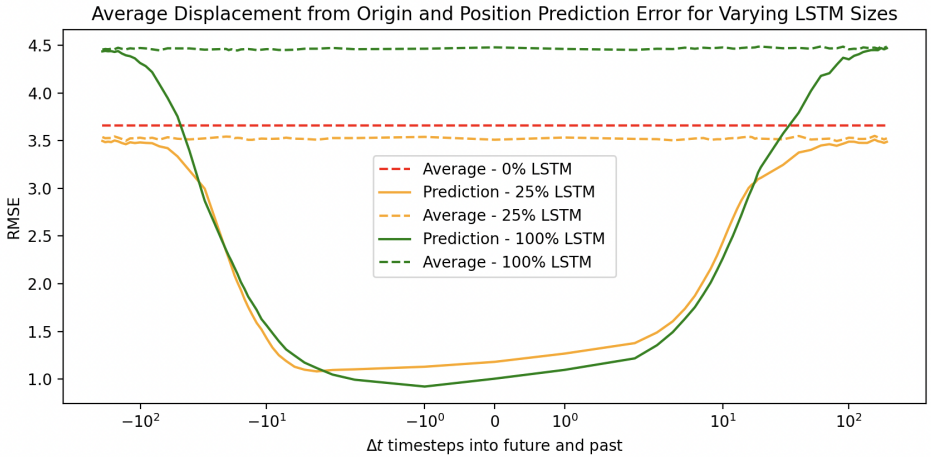



Figure 9: Comparing memory between different LSTM sizes. Also showing how far away from the center each agent is willing to move.

There are several aspects to consider in this plot. First, we discuss the width of the prediction error curves, which describes how long the memory reaches into the past and future. We see that the 25% LSTM agent is able to predict its position better than average displacement from origin for $|\Delta t| < 100$ timesteps. The average displacement from the origin (dashed lines) are essentially naive predictors, predicting the center every time. It therefore makes sense that for large $\Delta t$ the lines for Average and Prediction meets. For the 100% LSTM agent the memory-reach is closer to 150, indicating that it is able to remember further into the past. This is not surprising, as its memory storage is four times larger.

Second, we see that the average displacement of the 100% LSTM agent is much larger than the two other. This is also visualized previously where the agent moves further away from the tree when it tries to get rid of the predator. However, it is also interesting that the 0% LSTM agent and the 25% LSTM agent travel the same distance from the tree on average. This is most likely because the 25% LSTM agent has found a policy where it can stay close to the tree, and use its memory for other things - for example the position of the predator.

Third, we see that the 100% agent has the lowest prediction error for its current position. The RMSE is just over 1.0 which means that it on average is able to guess its own position based on the activation in the hidden states with an error of just one tile. What makes this even more impressive, is that it moves further from the tree than the other agents which would make it harder to predict its position.

Based on this plot, we can see (maybe not surprisingly), that the 100% LSTM agent has superior memory over the agent with less LSTM neurons.

### 3.4 Additional experiments

We also tested other environment combinations than the ones presented herein. For example, we tested a small environment with multiple predators. This was not very interesting, as the high risk of getting eaten forced the agent into the local optimum of ignoring apples completely and hiding in a corner.

We also tested multiple apple trees, and multiple predators. It was interesting to observe the agent learn to navigate the environment, but it was hard to analyze the behavior of the agent. Generally, the LSTM agent performed the best, but when evaluating the policy it was not always easy to see how the memory was utilized by the agent to improve performance.

We also ran tests without hunger as part of the input state and with a learning rate scheduler decreasing over time, but neither gave any insightful results or changes in performance.

## 4 Conclusion

In this project, we have successfully demonstrated the application of reinforcement learning combined with LSTM networks to enable memory-driven navigation in predator-prey environments. We developed a custom 2D grid-world simulation and used PPO to train agents capable of recalling spatial information without explicit positional awareness.

Our experiments show that agents equipped with LSTM significantly outperform those without, particularly in complex environments where memory provides a strategic advantage. The LSTM-enabled agents demonstrate advanced behaviors, such as revisiting critical areas and avoiding predators, effectively encoding spatial and temporal dependencies. Memory analysis confirm the agents' ability to predict positions over time, with larger LSTM architectures showing superior performance.

These findings highlight the potential of memory mechanisms in reinforcement learning to enhance decision-making in dynamic environments, inviting further exploration of their use in more advanced and diverse settings.

## References


[1] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory, 2024.

[2] Maciej Grzelczak and Piotr Duch. Deep reinforcement learning algorithms for path planning domain in grid-like environment. *Applied Sciences*, 11(23):11335, 2021.

[3] S Hochreiter. Long short-term memory. *Neural Computation MIT-Press*, 1997.

[4] Phone Thiha Kyaw, Aung Paing, Theint Theint Thu, et al. Coverage path planning for decomposition reconfigurable grid-maps using deep reinforcement learning based travelling salesman problem. *IEEE Access*, 8:225945–225957, 2020.

[5] Phone Thiha Kyaw, Aung Paing, Theint Theint Thu, Rajesh Elara Mohan, Anh Vu Le, and Prabakaran Veerajagadheswar. Coverage path planning for decomposition reconfigurable grid-maps using deep reinforcement learning based travelling salesman problem. *IEEE Access*, 8:225945–225957, 2020.

[6] Xiaoyun Lei, Zhian Zhang, and Peifang Dong. Dynamic path planning of unknown environment based on deep reinforcement learning. *Journal of Robotics*, 2018:Article ID 5781591, 2018.

[7] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation, 2019.

[8] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. `https://arxiv.org/pdf/1506.02438`.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. `https://arxiv.org/pdf/1707.06347`.

[10] YungMin SunWoo and WonChang Lee. Comparison of deep reinforcement learning algorithms: Path search in grid world. In *International Conference on Electronics, Information, and Communication (ICEIC)*, pages 20–21. IEEE, 2021.

[11] Hamid R. Tizhoosh. Reinforcement learning based on actions and opposite actions. *AIML Conference Proceedings*, 2005.

[12] Hamid R. Tizhoosh. Reinforcement learning based on actions and opposite actions. *AIML Conference Proceedings*, 2005.

[13] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke. The microsoft 2017 conversational speech recognition system. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5934–5938, 2018.