Air Traffic Control as a Deep Reinforcement Learning Problem

Zain Abbas, Daana Masumi, Peter Nadel

December 2025

1 Introduction

Air Traffic Control (ATC) is a high-stakes decision-making problem in which controllers must ensure safe separation between aircraft while optimizing operational efficiency. Worldwide passenger traffic is projected to double to 8.2 billion by 2037 [1] increasing the complexity and density of managed airspace. As global air traffic continues to grow, the need for intelligent automation becomes increasingly important. Deep Reinforcement Learning (DRL) offers a promising approach to ATC automation by enabling agents to learn control policies from high-dimensional continuous state spaces. However, existing DRL approaches for ATC tasks often struggle with long-horizon temporal dependencies inherent in aircraft trajectory planning and conflict resolution. In this work, we investigate whether incorporating memory through recurrent architectures improves learning across diverse ATC environments. Using the BlueSky-Gym simulation platform, we compare standard Proximal Policy Optimization (PPO) against RecurrentPPO with LSTM layers on seven benchmark environments, ranging from simple vertical descent to complex multi-aircraft merging scenarios.

2 Background and Related Works

2.1 Deep Reinforcement Learning for ATC

DRL has emerged as a promising approach for automating ATC tasks. Razzaghi et al. provides a survey of RL applications in aviation, highlighting the growing interest in applying DRL methods to conflict resolution and multi-agent navigation [7]. Wang et al. reviews deep reinforcement learning specifically for conflict resolution in ATC, examining various algorithmic approaches and their applications to collision avoidance scenarios [11].

Traditional approaches to ATC automation rely on rule-based systems and classical optimization methods, which struggle with scaling and adapting in dynamic environments. More recent approaches leverage Deep Q-Networks to handle the complexity, but it is hard to compare results across studies as they are tested on custom simulators with varied assumptions. Lack of standardized benchmark environments has made it difficult to objectively compare different methods across studies.

2.2 Recurrent Architectures in Reinforcement Learning

Recurrent neural networks, particularly Long Short-Term Memory (LSTM) networks, are effective in domains requiring memory of past states. Hausknecht and Stone demonstrated that Deep Recurrent Q-Networks (DRQN) enable agents to handle partial observability by integrating information over time [3]. Their work showed that even when seeing only a single frame per timestep, DRQN successfully learned to play Atari games by maintaining hidden states that capture temporal dependencies. Building on these insights, recurrent architectures have been integrated with policy gradient methods, allowing agents to learn policies that leverage temporal information for sequential decision-making tasks [4].

The combination of LSTMs with Proximal Policy Optimization, known as RecurrentPPO, extends PPO's stable on-policy learning with the temporal modeling capabilities. This architecture is implemented in Stable-Baselines3 library [6]. Stable-Baselines3 is a reliable open-source framework designed for ease of use with extensive documentation.

Groot et al. introduced BlueSky-Gym, a standardized benchmark platform built on the BlueSky air traffic simulator and the Gymnasium API [2]. Their initial experiments compared PPO, SAC, DDPG, and TD3 across seven ATC environments: DescentEnv, VerticalEnv, PlanWayPointEnv, HorizontalCREnv, SectorCREnv, StaticObstacleEnv, MergeEnv. Our work extends this foundation by investigating whether adding recurrent architectures to PPO improves learning on these same benchmarks, gaining insight into the value of temporal memory for ATC.

3 Technical Background

3.1 Reinforcement Learning Framework

In our ATC environment, we model the problem as a single decision-making agent, where the agent observes airspace state and issues control actions that influence aircraft trajectories. Using the BlueSky Gym environments, the task is modeled as a POMDP (Partial Observable Markov Decision Process), where the agent receives partial observations of the airspace environment and must make decisions based on incomplete or uncertain state information. In a typical episode, the agent observes the environment, takes actions based on policy, receives rewards at each step until the episode terminates [7].

3.2 Proximal Policy Optimization (PPO)

In a typical aircraft collision avoidance, PPO algorithm has shown promising results. It is an on-policy method based on the actor-critic framework. In PPO, the actor represents a stochastic policy that maps observations to actions, while the critic estimates the value functions used to compute advantage estimates for policy updates. The algorithm generates state transitions using a fixed policy. The state transitions are then used to update the policy directly.[9].

Algorithm 1 PPO-Clip [5]

```
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, ...$ do
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
6:     Update the policy by maximizing the PPO-Clip objective:
       $$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$
       typically via stochastic gradient ascent with Adam.
7:     Fit value function by regression on mean-squared error:
       $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$
       typically via some gradient descent algorithm.
8: end for
```

Traditional policy gradient methods are prone to unstable learning due to large policy updates. If the step size of the gradient is too small the training process was too slow to reach optimal policy. If the step size is too high, there was too much variability in the training. PPO addresses this issue by constraining policy updates to remain within a trust region. It often uses objective function called Clipped surrogate objective function that constrains the policy change in a small range using a clip.

PPO performs its clipped surrogate function by calculating the ratio between the probability of action under current policy divided by the likelihood of the action under the previous policy defined as $rt(Q)$. If $rt(Q) > 1$ then the action is more probable in the current policy than the old one. If $rt(Q)$ is between 0 and 1, the action is less probable to the current policy than the old one. Without adding a constraint, the action taken is more probable in our current policy than former, leading to significant policy gradient step and excessive policy update. The clipping mechanism essentially limits the deviation between the new and old policies, preventing excessively larges updates while maintaining efficient learning. The clipping mechanism restricts the probability ratio to $1-\epsilon \leq rt(\theta) \leq 1+\epsilon$. The PPO takes the minimum of the clipped and unclipped function, so the final objective is a lower bound on the unclipped objective [9].

In the ATC environment, PPO helps the agent learn conservatively in partial observable environment. The agent learns continuous control laws, avoids drastic policy shifts, and improves decision-making incrementally.

3.3 BlueSky Gym Environments

To test our agent in the ATC environment, we used the open-source air traffic simulator called BlueSky-Gym. All the environments in BlueSky-Gym are derived from Gymnasium's ENV class, which are based on Markov Decision Process (MDP) logic. Typical RL models are trained on a fixed set of scenarios, where the models memorize a specific pattern. This method prevents the model from generalizing to a broader subtask. By leveraging the BlueSky-Gym package, we train our model in a continuous environment that prevents overfitting on repeated samples. Thus, the rewards gained in a continuous environment ensure that it correctly attributes to properly trained policies [2].

Typical users train RL agents on self-defined model or use Stable-Baselines3 model to train and test the performance of the model. In our case, we utilized Stable-Baselines3 model called PPO along with LSTM. Once the model is designed, we trained our agent using BlueSky environments [2].

3.3.1 SectorCREnv-0

In SectorCREnv environment, an agent is trained to exit the airspace sector as fast as possible, while avoiding moving obstacles. Th figure above shows an agent surrounded by moving obstacles. The white boundary defines the airspace sector in which the agent is spawned.

3.3.2 StaticObstacleEnv-v0

This environment is based on HorizontalCR environment, where the agent must learn to navigate in a horizontal environment while avoiding static obstacles. This type of environment has practical applications in ATM (Air Traffic Management) and UTM (Unmanned Traffic Management). In ATM, the obstacles represent restricted airspace whereas, in UTM, they represent trees, buildings and other big obstacles that agents must avoid.

3.3.3 DescentEnv-v0

In this environment, the agent is learning to control vertical descent by optimizing the speed of the aircraft. The key goal is to maintain random target

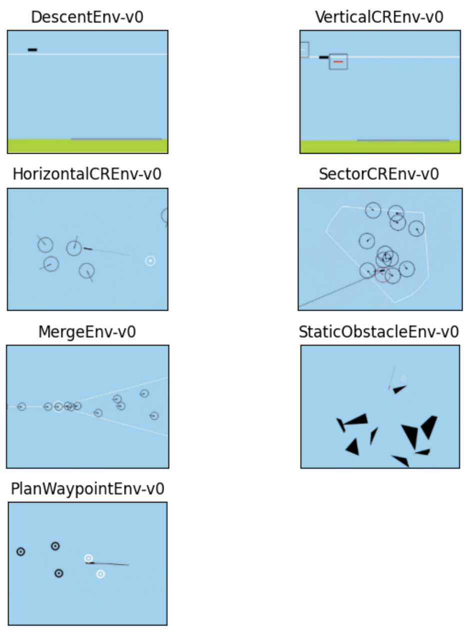

Figure 1: All of the BlueSky-Gym environments

altitude before descending to the runway.

3.3.4 VerticalCREnv-v0

The VerticalCREnv focuses on conflict resolution environment. This environment has the same structure as the DescentEnv, focusing on altitude and runway. In addition, it includes cruising aircraft with conflicting trajectories that should also be avoided by the target.

3.3.5 PlanWaypointEnv-v0

In the PlanWaypoint environment, the agent must plan efficiently to navigate in a horizontal framework. Given a waypoint observation vector relative to agent's location, the agent must design a trajectory to visit randomly generated waypoints.

3.3.6 HorizontalCREnv-v0

Based on the PlanWaypointEnv-v0, the agent learns to traverse to destination while avoiding other moving aircraft. Random aircraft are generated in the vicinity of the agent's aircraft. This specific environment has practical applications in ATM and UTM.

Environment	Actions	Observations	Reward	Goal
DescentEnv-v0	vertical speed	4	Dense	Hold target altitude, land on time
VerticalCREnv-v0	vertical speed	$4 + 7 \times n_{ac}$	Dense, Sparse	Hold target altitude, land on time, avoid conflicting aircraft
PlanWaypointEnv-v0	heading	$4 \times n_{wpt}$	Sparse	Visit all waypoints once
HorizontalCREnv-v0	heading	$3 + 5 \times n_{ac}$	Dense, Sparse	Get to waypoint, avoid conflicting aircraft
SectorCREnv-v0	heading, speed	$3 + 7 \times n_{ac}$	Dense, Sparse	Get out of sector, avoid conflicting aircraft
StaticObstacleEnv-v0	heading, speed	$3 + 4 \times n_{obs}$	Sparse	Get to waypoint, avoid static obstacles
MergeEnv-v0	heading, speed	$5 + 7 \times n_{ac}$	Dense, Sparse	Merge into traffic, avoid aircraft, reach waypoint

Table 1: Summary of all environments

3.3.7 MergeEnv-v0

MergeEnv-v0 is one of the unique environments within the BlueSky framework. In this environment, a large group of aircraft fly toward a common waypoint and then continue toward a runway. The agent must learn to navigate safely by avoiding collisions with nearby aircraft while proceeding toward the shared waypoint and ultimately passing through the runway. This environment has practical applications in ATM and UTM, particularly for coordinating multiple aircraft to land sequentially while merging without conflict.

3.4 RecurrentPPO and Long Short-Term Memory

PPO employs frame-stacking to carry over key pieces of information from one episode to the next [9]. Because a single step in an episode does not provide all of the information for the agent to make a decision, environments in which PPO is often deployed are non-Markovian, breaking the assumption of a Markov Decision Process (MDP) that an agent's future actions depend only on the current state. Frame-stacking can help in this regard. Though not much more complex than concatenating the last k observations into a single input, this strategy is effective in approximating an MDP for environments where it would be impossible otherwise. However, in this project, we sought to elaborate on standard PPO by replacing this frame-stacking step with a more complex alternative. In recurrent PPO, rather than the past k timesteps being stacked and fed into the feature extractor as an input, we use a recurrent neural network (RNN) to extract latent features from past time steps. RNNs have proven widely exploitable outside of reinforcement learning, particularly in the spaces of computer vision (CV) and natural language processing (NLP), but recurrent architectures are less obviously valuable in reinforcement learning contexts [8]. This difference arises for the following reasons: First, RNNs, though they are able to produce powerful latent features in CV and NLP, take up a much large footprint to train. They are generally larger and require deep networks to be successful, whereas frame-stacking is computationally simpler and therefore potentially more efficient. Next, especially in the case of NLP, RNNs excel in partial observable environments. Consider an autoregressive or masked language modeling task as an example. The RNN must learn the latent representation of tokens stored in its learned memory to complete or fill in text sequences. This case is not always true in RL, where environments can be fully or mostly observable, though still not pure MPDs, and where additional data processing could be a hindrance rather than a help [8].

Algorithm 2 Recurrent PPO-Clip

```
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, ...$ do
3:     Initialize environment and hidden state $h_t$ to 0
4:     Collect trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
5:     Store collected trajectories data (states, actions, rewards, terminal states, log probabilities and previous hidden states) as ($s_k$, $a_k$, $r_k$, $d_k$, $\log(\pi_{\theta_k}(a_k|s_k, h_{k-1}, h_{k-1})))$, where $d$ is a boolean value which would indicates when the reset the hidden state.
6:     Compute rewards-to-go $\hat{R}_t$.
7:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
8:     for $e = 0, 1, 2, ...$ do
9:         Split trajectories $\mathcal{D}_k$ into mini-batches.
10:        for $b = 0, 1, 2, ...$ do
11:            Propagate observations in $b$ through LSTM
12:            Update the policy by maximizing the PPO-Clip objective:
               $$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$
               typically via stochastic gradient ascent with Adam.
13:            Fit value function by regression on mean-squared error:
               $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$
               typically via some gradient descent algorithm.
14:        end for
15:    end for
16: end for
```

That said, we affirm that in the ATC context, there are partially observable situations in which extra recurrent processing can augment agent performance. To that end, we implement a simple recurrent PPO algorithm with the Long Short Term Memory (LSTM) architecture. Though introduced much earlier, LSTMs came to prominence in the mid 2010s for classification and prediction tasks, especially in NLP, where it was used by Google for voice recognition, machine translation and auto-completion [10]. Too, both OpenAI and Google DeepMind employed LSTMs for training agents to beat humans in video games, Dota 2 and Starcraft II, respectively.

These applications took advantage of the LSTM's ability to remember relevant information and forget irrelevant information. This architecture is designed to learn which information might be helpful later on in the given sequence. A classical problem in constructing deep neural networks is when certain gradients, over several thousand iterations of back-propagation, approach zero, effectively rendering many, if not a large majority of neurons, useless. LSTM seeks to solve this so-called "vanishing gradients" problem by learning gate value parameters between zero and one. Because these values are much more predictable in their behavior and can be aggregated with element-wise operations, there are less opportunities for vanishing gradients. Specifically, LSTM layers implement four feature gates: forget, input, output and cell. Each of these parameters can be much more closely controlled and have much fewer possible values, allowing for stabler and more consistent training. That said, because there are more parameters in the LSTM, especially compared to the frame-stacking PPO case, this architecture is more susceptible to over-fitting, particularly in situations where there is a small amount of training data [10].

4 Experimental Design

As we describe above, the PPO learning algorithm excels in training agents in continuous environments when policy networks get very large. In the original publication in which BlueSky-Gym was introduced, the authors use several off-the-shelf implementations of various learning algorithms (PPO, DDPG, TD3, SAC), but PPO consistently provided useful results, although the performance of a particular algorithm varied significantly from environment to environment [2]. As a result, we chose PPO as a baseline, against which to compare algorithms which introduce recurrent processing.

Building on this baseline, then, recurrent PPO employs a set of LSTM-based neural networks to understand past actions in an episode and use this extra information to inform policy updates, in the form of latent feature representations. With this experiment, we seek to understand the extent to which each of the BlueSky ATC environments benefits from the addition of recurrent architectures. Indeed, just because recurrent PPO is a more complex architecture than standard PPO, this does not mean that recurrent PPO is better in all scenarios. In fact, we hypothesize that recurrent PPO will only outperform standard

PPO in cases where the environment is not fully observable to the agent, as, in these cases, the agent must rely on a spatial memory to supplement its observations, while in more observable environments, this extra memory would only add information that would be conflicting or confusing. Thus, this experiment will show which of the key ATC tasks identified by BlueSky-Gym could benefit from recurrent learning and to what extent this addition would result in better outcomes, as measured by mean reward per episode and mean episode length.

To better understand the effect of recurrent PPO, we chose to compare two different variants of the algorithm to standard PPO: one where both critic and actor networks are 2-layer LSTM neural networks and another where they are 4-layer LSTM neural networks. Both of these variants (2-layer and 4-layer), as well as the standard PPO network, employ a hidden state size of 256 parameters. Testing recurrent PPO at these different sizes will help us grasp the practical effects that it has on policy training. For example, if we observe that 4-layer recurrent PPO outperforms the 2-layer variant, we may conclude that this environment benefits from the recurrent algorithm, but requires more and deeper feature extraction to make this algorithm truly effective. If the opposite is observed, that the 2-layer variant bests the 4-layer one, we may infer that the extra LSTM layers are not needed and instead add noise to the output features, leading to worse performance overall. Too, we may observe that standard PPO performs better than both the 2-layer and 4-layer LSTM agents for a given environment. In these cases, we gather that this environment is observable enough such that the agent does not require the extra features provided by the recurrent architecture.

In the same vein as the BlueSky-Gym authors, who claim "the environments have been designed such that adequate results and stability can be obtained with minimal hyperparameter tuning," we did not vary and hyperparameters and maintained the ones mentioned above across all environments [2]. Additionally, we replicated their standard PPO results with a learning rate of 0.0003, whereas we found the best recurrent PPO results with a slightly lower value of 0.0005. Besides this, we trained our networks for one million timesteps, only half of what the BlueSky-Gym authors did. We felt that this training length would be sufficient to examine our primary interest, a direct comparison of standard PPO and recurrent PPO.

[Algorithms] [Tables]

5 Experimental Results

The experimental results from training Standard PPO and RecurrentPPO variants across the seven BlueSky-Gym environments over one million timesteps are presented below. Performance is evaluated based on mean episode reward and mean episode length, where higher rewards indicate better performance.

In SectorCREnv-v0, Standard PPO achieved a shorter average episode length

Environment	Standard PPO	2-Layer LSTM	4-Layer LSTM
SectorCREnv-v0	-19.1	-22	-22.7
DescentEnv-v0	-24.2	-43.2	-40.4
HorizontalCREnv-v0	-1.9	-3.8	-3.6
MergeEnv-v0	-0.7	-1.8	-1.5
PlanWaypointEnv-v0	4.8	4.1	4.2
StaticObstacleEnv-v0	-3.4	-3.1	-2.6
VerticalCREnv-v0	-119.7	-114.1	-124.4

Table 2: Mean reward per episode at the end of training

Environment	Standard PPO	2-Layer LSTM	4-Layer LSTM
SectorCREnv-v0	64.5	52.9	65.3
DescentEnv-v0	40.0	39.3	39.5
HorizontalCREnv-v0	26.7	33.2	31.5
MergeEnv-v0	29.3	25.8	31.3
PlanWaypointEnv-v0	231.0	251.3	263.7
StaticObstacleEnv-v0	53.4	65.6	70.0
VerticalCREnv-v0	38.4	38.5	10.4

Table 3: Mean episode length at the end of training

(64.5 steps) compared to RecurrentPPO with 4 LSTM layers (65.3 steps) while simultaneously achieving superior rewards (-19.11 vs -22.71). This demonstrates that Standard PPO navigates the sector more efficiently, finding faster exit routes while avoiding conflicts. The longer episodes of the 4-layer variant suggest it takes more cautious, indirect paths that accumulate additional time-based penalties without providing corresponding safety benefits.

In DescentEnv-v0, all three algorithms achieved nearly identical episode lengths ranging from 39 to 40 steps, indicating successful task completion without premature termination. However, Standard PPO achieved significantly better average reward of -24.2 compared to RecurrentPPO variants. This performance gap reveals that while all algorithms complete the descent, Standard PPO maintains the target altitude more precisely throughout the episode. The recurrent variants accumulate nearly double the penalties, suggesting their LSTM layers introduce instability or oscillations in altitude control that increase deviation from the target.

In HorizontalCREnv-v0, Standard PPO achieved the shortest episode length (26.7 steps) while simultaneously achieving the best rewards (-1.9) compared to RecurrentPPO variants. This demonstrates that Standard PPO navigates to the waypoint most efficiently, finding direct paths that avoid conflicts with minimal time penalties. The recurrent variants take longer to complete the task while accumulating nearly double the penalties, suggesting their LSTM layers introduce overly cautious behavior that leads to inefficient routes without

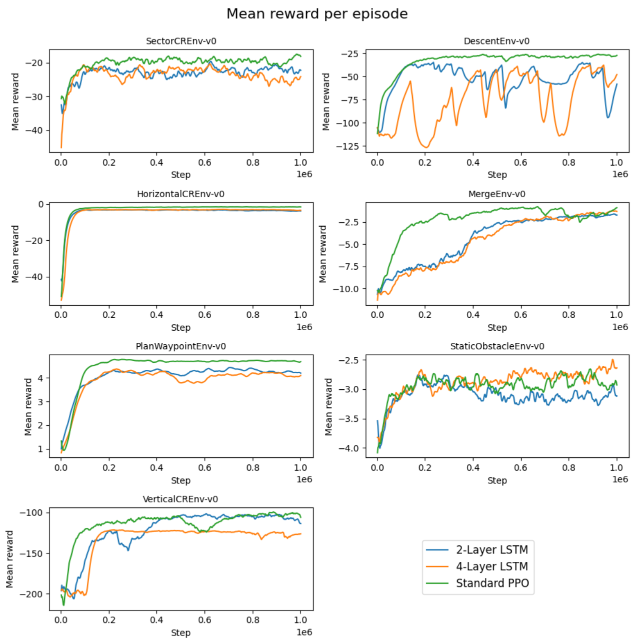

Figure 2: Mean reward per episode

providing corresponding safety benefits.

In MergeEnv-v0, Standard PPO achieved the best performance with an average reward of -0.7 and average episode length of 29.3 steps, demonstrating smooth integration into traffic flow while avoiding conflicts. The 2-layer RecurrentPPO exhibited the shortest episodes (25.8 steps) but worst rewards (-1.8), indicating it crashes or experiences conflicts early during the complex merge maneuver. Conversely, the 4-layer variant took slightly longer (31.3 steps) with moderate rewards (-1.5), suggesting overly cautious behavior. Standard PPO's superior performance reveals it has learned the optimal balance between decisive merging and safety, while the recurrent variants either act too aggressively (2-layer) or too hesitantly (4-layer).

In PlanWaypointEnv-v0, Standard PPO achieved the best average reward of 4.8 while completing the task in the shortest time of 231.0 average steps per episode. This demonstrates that Standard PPO plans the most efficient trajectories to visit all waypoints, completing the task 9-14% faster while earning more rewards. The recurrent variants take significantly longer to visit the same waypoints, suggesting their LSTM layers introduce suboptimal pathfinding decisions or hesitation that delays waypoint visitation. In this sparse reward environment where rewards are only given upon reaching waypoints, the longer episodes combined with lower total rewards indicate the recurrent architectures are less efficient at planning optimal routes through the waypoint sequence.

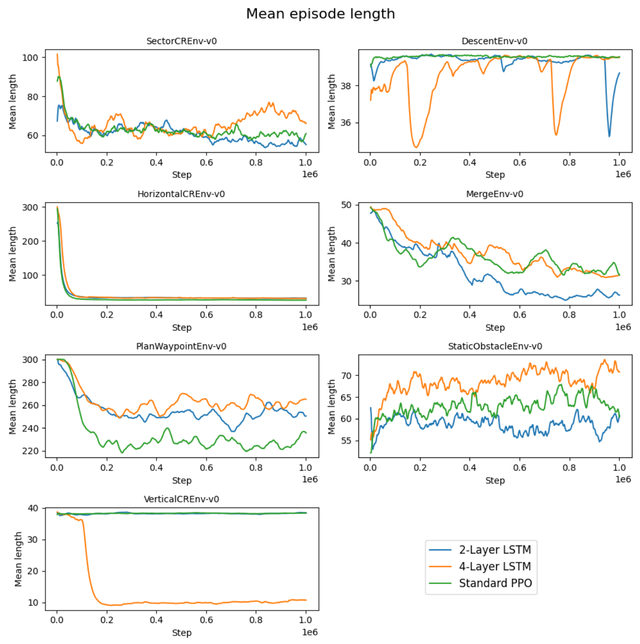

Figure 3: Mean episode length

In VerticalCREnv-v0, the RecurrentPPO with 2 LSTM layers achieved the best average reward of −114.1 outperforming standard PPO (-119.7). They had similar episode lengths around 38-39 steps, showing successful task completion without premature termination. This indicates that both these algorithms instructed the agent to maintain vertical control, safe descent, and separation from conflicting aircraft. The RecurrentPPO with 4 layers suffered an average of 10.4 steps and an average reward of −124.4.

StaticObstacleEnv-v0 represents a notable exception to the overall trend, as both RecurrentPPO variants outperformed Standard PPO. RecurrentPPO with 4 LSTM layers achieved the best performance (-2.6 reward) with the longest episode length of 70 steps, showing successful navigation around static obstacles. The 2-layer variant showed intermediate performance (-3.1 at 65.6 steps), while Standard PPO achieved the worst rewards (-3.4) with the shortest episodes (53.4 steps). The positive correlation between episode length and reward shows that the longer episodes mean better obstacle avoidance. Agent that navigate carefully around obstacles stay in flight longer and accumulate fewer collision penalties. This represents one of the rare environments where LSTM memory provides clear benefits, as remembering spatial layout of static obstacles enables better path planning and collision avoidance compared to the memoryless approach.

6 Conclusion and Future Work

6.1 Conclusion

This study investigated whether incorporating memory through recurrent architectures improves reinforcement learning performance across Air Traffic Control environments. Our results largely confirm our initial hypothesis that recurrent PPO benefits only partially observable environments. Standard PPO outperformed RecurrentPPO variants on five of seven BlueSky-Gym environments (SectorCR, DescentEnv, HorizontalCREnv, MergeEnv, and PlanWaypointEnv), while one or more variants of RecurrentPPO excelled only on StaticObstacleEnv and VerticalCREnv. The key finding is that LSTM memory provides benefits only when spatial layouts or temporal patterns must be remembered. In fully observable environments where current observations contain complete state information, recurrent architectures degrade performance through training difficulties. The failure of 4-layer LSTM on VerticalCREnv shows that large memory can lead to serious optimization issues. The results illustrate that the neural network architectures for ATC automation should match the observability characteristics of the task. For standard ATC scenarios with full observability, simpler PPO policy is the best choice. Recurrent architectures should be used in scenarios where memory of spatial layouts provides advantages.

6.2 Future Work

There are a few approaches to extend this research. Training duration could be increased from 1 million to 2 million timesteps to determine whether RecurrentPPO's underperformance reflects fundamental limitations or slower convergence. Alternative recurrent architectures such as attention mechanisms could be tested to reveal whether the observed limitations are specific to LSTM or inherent to recurrent processing in fully observable tasks. Another improvement would be evaluating on scenarios with genuine partial observability such as limited radar coverage, to identify environments where recurrent architectures might demonstrate clearer advantages. Finally, extending comparisons to off-policy recurrent algorithms such as Recurrent SAC, would determine whether limited memory benefits are specific to on-policy methods or represent a general phenomenon in RL-based ATC automation.

7 Acknowledgments

We would like to acknowledge the Tufts University High Performance Compute Cluster which was utilized for the research reported in this project.

References

[1] International Air Transport Association. *Aviation Ground Handling.* 2025. URL: `https://www.iata.org/en/training/pages/aviation-ground-handling-report/`.

[2] D. J. Groot et al. “BlueSky-Gym: Reinforcement Learning Environments for Air Traffic Applications”. In: *SESAR Innovation Days 2024.* 2024. URL: `repository.tudelft.nl`.

[3] Matthew Hausknecht and Peter Stone. *Deep Recurrent Q-Learning for Partially Observable MDPs.* 2017. arXiv: `1507.06527 [cs.LG]`. URL: `https://arxiv.org/abs/1507.06527`.

[4] Nicolas Heess et al. *Memory-based control with recurrent neural networks.* 2015. arXiv: `1512.04455 [cs.LG]`. URL: `https://arxiv.org/abs/1512.04455`.

[5] OpenAI. *Proximal Policy Optimization.* 2018. URL: `https://spinningup.openai.com/en/latest/algorithms/ppo.html`.

[6] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[7] Pouria Razzaghi et al. “A survey on reinforcement learning in aviation applications”. In: *Engineering Applications of Artificial Intelligence* 136 (Oct. 2024), p. 108911. ISSN: 0952-1976. DOI: `10.1016/j.engappai.2024.108911`. URL: `http://dx.doi.org/10.1016/j.engappai.2024.108911`.

[8] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview.* 2019. arXiv: `1912.05911 [cs.LG]`. URL: `https://arxiv.org/abs/1912.05911`.

[9] John Schulman et al. *Proximal Policy Optimization Algorithms.* 2017. arXiv: `1707.06347 [cs.LG]`. URL: `https://arxiv.org/abs/1707.06347`.

[10] Christian Bakke Vennerød, Adrian Kjærran, and Erling Stray Bugge. *Long Short-term Memory RNN.* 2021. arXiv: `2105.06756 [cs.LG]`. URL: `https://arxiv.org/abs/2105.06756`.

[11] Zhuang Wang et al. “Review of Deep Reinforcement Learning Approaches for Conflict Resolution in Air Traffic Control”. In: *Aerospace* 9.6 (2022). ISSN: 2226-4310. DOI: `10.3390/aerospace9060294`. URL: `https://www.mdpi.com/2226-4310/9/6/294`.