



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**APPLIED REINFORCEMENT LEARNING WARGAMING
WITH PARALLELISM, CLOUD INTEGRATION,
AND AI UNCERTAINTY**

by

Matthew G. Finley

June 2023

Thesis Advisor:
Second Reader:

Christian J. Darken
Armon C. Barton

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 2023	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE APPLIED REINFORCEMENT LEARNING WARGAMING WITH PARALLELISM, CLOUD INTEGRATION, AND AI UNCERTAINTY			5. FUNDING NUMBERS	
6. AUTHOR(S) Matthew G. Finley				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) The recent rapid growth in machine learning and artificial intelligence has created a growing opportunity to improve Department of Defense (DOD) wargaming. This study aims to leverage modern frameworks, algorithms, and cloud hardware to improve the DOD's wargaming capabilities, with a specific focus on reducing training times, increasing deployment flexibility, and demonstrating how a trained neural network can provide a measure of certainty for recommended actions. This work utilized an open-source framework for parallelization to train and deploy a neural network to the Azure cloud platform. To measure the certainty of a trained network's move choices, Bayesian variational inference techniques were employed. The application of the open-source framework resulted in a more than tenfold reduction in training times without any loss in performance. Additionally, deploying trained models to the Azure cloud platform effectively mitigated infrastructure constraints and Bayesian methods were successful in providing a measure of trained models certainty. The DOD could leverage these advancements in machine learning and cloud computing to significantly enhance future wargaming efforts.				
14. SUBJECT TERMS wargame, artificial intelligence, machine learning, cloud, parallelization, uncertainty, human machine teaming			15. NUMBER OF PAGES 69	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**APPLIED REINFORCEMENT LEARNING WARGAMING WITH
PARALLELISM, CLOUD INTEGRATION, AND AI UNCERTAINTY**

Matthew G. Finley
Lieutenant, United States Navy
BS, United States Naval Academy, 2016

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
June 2023**

Approved by: Christian J. Darken
Advisor

Armon C. Barton
Second Reader

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The recent rapid growth in machine learning and artificial intelligence has created a growing opportunity to improve Department of Defense (DOD) wargaming. This study aims to leverage modern frameworks, algorithms, and cloud hardware to improve the DOD's wargaming capabilities, with a specific focus on reducing training times, increasing deployment flexibility, and demonstrating how a trained neural network can provide a measure of certainty for recommended actions. This work utilized an open-source framework for parallelization to train and deploy a neural network to the Azure cloud platform. To measure the certainty of a trained network's move choices, Bayesian variational inference techniques were employed. The application of the open-source framework resulted in a more than tenfold reduction in training times without any loss in performance. Additionally, deploying trained models to the Azure cloud platform effectively mitigated infrastructure constraints and Bayesian methods were successful in providing a measure of trained models certainty. The DOD could leverage these advancements in machine learning and cloud computing to significantly enhance future wargaming efforts.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1	Introduction	1
1.1	Problem Framing	1
1.2	Scope and Structure of Thesis	1
1.3	Findings	2
1.4	Benefits of Thesis	2
2	Background	3
2.1	Requirements	3
2.2	Requirement Implementation	3
2.3	Algorithms	6
2.4	Thesis Reinforcement Learning Framework	7
3	Parallelization of Reinforcement Learning Applications	9
3.1	Problem Framing	9
3.2	Design of Experiment	9
3.3	PPO Results	11
3.4	IMPALA Results	14
3.5	Overall PPO vs IMPALA Comparison	19
3.6	Additional Parallelization Conclusions	20
4	Azure Cloud Integration	21
4.1	Problem Framing	21
4.2	Solution Overview	22
4.3	Environment Development	22
4.4	Initialization and Deployment	25
4.5	Benefits of Cloud Deployment	27
5	Bayesian Implementation	29
5.1	Problem Framing	29

5.2	Applying Bayesian Methods to Neural Networks	29
5.3	Experiments	33
5.4	Unexpected Scenarios	38
6	Conclusion	41
6.1	Benefits of Parallelization	41
6.2	Practically Deploying Reinforcement Learning Trained Models	42
6.3	Benefits of Bayesian Techniques in RL Models	42
6.4	Recommendations for Future Work	43
6.5	Summary and Recommendations	44
	Appendix: REQUEST RATE AND LATENCY LOG METRICS	45
	List of References	47
	Initial Distribution List	49

List of Figures

Figure 2.1	Example of a City-Inf-5 scenario	4
Figure 2.2	Illustration of hexagonal convolutions. Source: [4]	5
Figure 3.1	PPO vs CPU Training Time	11
Figure 3.2	PPO CPU + GPU Results (h)	12
Figure 3.3	PPO GPU overlaid over CPU only training time	13
Figure 3.4	PPO Best CPU only vs Best CPU + GPU	14
Figure 3.5	IMPALA vs CPU training time	15
Figure 3.6	PPO w/GPU Training Time	17
Figure 3.7	IMPALA CPU and CPU + GPU overlaid	18
Figure 3.8	IMPALA Best CPU only vs Best CPU + GPU	18
Figure 3.9	All time results by experiment	19
Figure 3.10	PPO and IMPALA Highest episode reward Mean vs Time Total .	20
Figure 4.1	Future relationships between wargame developers and users across cloud and local instances	21
Figure 4.2	Functioning networking and a data path for deployed inference containers	22
Figure 4.3	Docker code block for building deployment Docker container . .	24
Figure 4.4	Flexible image deployment options	25
Figure 4.5	Initialization and serving data paths within deployed inference containers	26
Figure 5.1	Illustration of a non-Bayesian (Deterministic) and a Bayesian (probabilistic) Neural Network. Source: [16]	31

Figure 5.2	Illustration of a Bayesian network with high and low certainty. . .	31
Figure 5.4	Graph of the mean reward during training. Non-Bayesian	34
Figure 5.3	Graph of the maximum reward during training. Non-Bayesian . .	35
Figure 5.5	Graph of the mean reward during training, Bayesian Model . . .	36
Figure 5.6	Graph of the maximum reward during training, Bayesian Model .	36
Figure 5.7	Change in epistemic uncertainly per move at four training iterations.	37
Figure 5.8	Graph of model uncertainty with additional city present.	38
Figure 5.9	Graph of model uncertainty with new red force unit present. . . .	38
Figure 6.1	PPO and IMPALA Highest episode reward Mean vs Time Total .	41
Figure 6.2	Change in epistemic uncertainly per move at four training iterations.	42
Figure A.1	Graph of deployed model latency and request rate	45

List of Tables

Table 2.1	The implemented non-Bayesian network	6
Table 3.1	Table of PPO and IMPALA Runs per number of assigned CPUs .	10
Table 3.2	Table of PPO CPU parallelization results	11
Table 3.3	PPO with CPU + GPU	12
Table 3.4	IMPALA CPU Results	15
Table 3.5	IMPALA CPU + GPU Results (h)	16
Table 3.6	All experiment timing results	19
Table 5.1	The implemented non-Bayesian network	34
Table 5.2	The implemented Bayesian network	35
Table 5.3	Epistemic uncertainty per time step after 10, 60, 1850, and 1900 training iterations.	37

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

AI	artificial intelligence
ASGI	Asynchronous Server Gateway Interface
DOD	Department of Defense
GPU	graphics processing unit
IMPALA	Importance Weighted Actor-Learner Architecture
IP	Internet Protocol
JSON	JavaScript Object Notation
ML	machine learning
PPO	Proximal Policy Optimization
RL	reinforcement learning
URL	Uniform Resource Locator

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

Artificial intelligence (AI) has witnessed remarkable advances over the past few decades. Recent advancements in deep learning and reinforcement learning (RL) have enabled AI models to exceed human performance in various video game formats. As the Department of Defense (DOD) continues to invest in the development of AI models for wargaming and warplanning applications, many improvements are available.

This research investigates the application of modern machine learning (ML) techniques to enhance the efficacy of wargaming simulations. This study demonstrates that parallelization can substantially reduce training times for RL problems, even in the absence of a graphics processing unit (GPU), with minimal impact on average scores. This finding underscores the importance of parallel processing frameworks for future RL training efforts. This research utilizes the Ray framework for orchestrating parallelization of RL training and evaluates two algorithms, Proximal Policy Optimization (PPO) and Importance Weighted Actor-Learner Architecture (IMPALA), both with and without GPU acceleration. This research successfully shows between one and two orders of magnitude in reduction of training time while maintaining overall average performance is easily achieved.

The second part of this research investigates a practical approach to decoupling locally trained models from local environments, showcasing the feasibility of deploying these models to cloud-based environments. The models employed were developed using open-source frameworks and deployed on the Microsoft Azure cloud platform. This research successfully deployed a trained RL model to the cloud environment and integrated into local training and evaluation.

Lastly, this thesis demonstrates that Bayesian techniques can be integrated within RL models, potentially enhancing the value of human-machine teaming. This is achieved by incorporating Bayesian methods into a model's architecture and leveraging the distinct attributes of these implemented layers during run-time. This research was successful and showed how a measure of certainty into an AI's move choice could be synthesized and presented to a human.

In summary, this research contributes to advancing ML and wargaming simulation tech-

niques by emphasizing the significance of parallelization, providing proof-of-concept of a trained model to a cloud-based environment, and demonstrating the feasibility of incorporating Bayesian methodologies into AI models to improve human-machine collaboration.

Acknowledgments

I would like to acknowledge the patience and trust Professor Darken gave me when I started this research. Without his support and advice, I would not have been able to address such a large range of topics. Additionally, none of this research would have been possible without the rest of the Computer Science faculty being so generous with their time and guidance.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

1.1 Problem Framing

Recent and rapid developments in machine learning (ML) hardware, learning techniques, and open frameworks have enabled the development of increasingly complex computational programs to play (or solve) wargames. As complexity continues to increase, the computational time required to train more sophisticated models will also likely increase. Optimizations are needed to move artificial intelligence (AI) enabled wargaming capabilities forward economically. Additionally, as industry and government become increasingly agile, more modularity is required to shift from proof-of-concept products to something that can be concurrently operated and developed. I posit that as government and industry become more cloud-centric in their capabilities, so will wargames and war planning applications. Lastly, human-machine teaming is the future of military planning and operations, but connecting the speed of AI execution to human intuition and creativity needs improvement.

1.2 Scope and Structure of Thesis

This thesis will explore how the developing concepts and technologies outlined in section 1.1 can be utilized to improve training rates significantly, show how a trained reinforcement learning (RL) model can be deployed to a cloud environment, and functionally illustrate how Bayesian methods can make new types of information available to humans to improve human-machine-teaming. I will investigate three distinct subtopics:

1. How can the speed of training wargame AI models with RL be improved through parallelization with and without a graphics processing unit (GPU)?
2. How can a trained model be deployed to a cloud or local environment without porting the wargame in its entirety?
3. Can modern Bayesian techniques expand the information provided to a human observer when an AI model is progressing through a wargame scenario?

Each of these topics will receive a dedicated chapter.

1.3 Findings

Concerning the items above;

1. A speed increase between one and two orders of magnitude can be achieved with little adaptation of a wargame environment and the application of open-source training algorithms and associated parallelization frameworks. This is before neural net architecture improvements or parameter tuning.
2. A model trained with open-source frameworks can successfully be “bottled,” deployed, and queried. Deployments can occur locally or within a cloud environment. All deployments leverage industry-standard tools and techniques.
3. Applying Bayesian techniques can provide a measure of a model’s certainty of its next action. Additionally, a AI model can be trained to calculate epistemic uncertainty and be shown, over training iterations, that it is converging to “optimality” based on trained experience.

1.4 Benefits of Thesis

This thesis is of direct benefit to the U.S. Government. Building autonomous and interconnected systems is increasingly essential to the Department of Defense (DOD). This thesis shows, demonstrably, how ML models for wargames may be trained an order of magnitude faster, increasing the iteration rate, and provides a method to deploy these trained models to improve the consumption rate and utility. Additionally, showing how Bayesian methods may be integrated helps provide a link between human and machine operators. It could significantly boost human-machine teaming and increase trust in remote or autonomous systems.

CHAPTER 2: Background

2.1 Requirements

Training and evaluation of a RL algorithm requires several elements,

- A training environment.
- An artificial neural network to implement into a decision policy.
- An algorithm to train a learned policy.

2.2 Requirement Implementation

2.2.1 Training Environment

This paper utilizes a training environment named Atlatl [1], [2]. In summation, Atlatl is a deep RL framework for a two-player turn-based wargame built on a client-server architecture. It uses Python, JavaScript, SVG, and the Stable Baselines3 library for its implementation. The game environment features a hexagonal grid map with various terrain and unit types. Players aim to maximize their score while minimizing their opponent's score, and combat outcomes follow a deterministic Lanchester model.

The game's state representation is transformed into an OpenAI Gym-conform observation space using a double coordinate system, and the neural network processes multiple feature inputs. Atlatl leverages deep RL techniques to train AI agents, making it an ideal framework for studying wargame simulations and reinforcement learning.

Atlatl simulates a simple wargame wherein two opposing players (usually AI players) take turns moving their respective pieces and attacking opponent pieces. Games last for several rounds, and scores can be tuned to various goals.¹ Previous work with and source descriptions of Atlatl can be found in [1] and [2].

¹Summary of [1] and [2] generated with ChatGPT, 18 May 2023, and rephrased for content and style.

All training runs in this thesis were done in a City-Inf-5 scenario. City-inf-5 is a simulation game mode involving a random number (between two and five) of red and blue units each, the unit types are all identical. The game is centered around a "city" tile that provides bonus reward points and score to the faction that holds it. The faction with fewer units has the city spawned with closer proximity, and the game lasts for ten turns. Figure 2.1 illustrates an example of City-Inf-5.

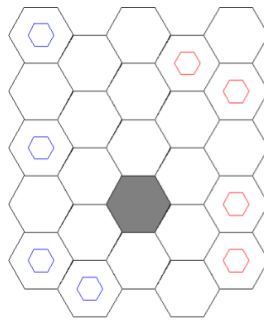


Figure 2.1. Example of a City-Inf-5 scenario

2.2.2 Artificial Neural Networks

The networks used in this thesis are based on the work done in [1] and [3]. In general, the networks consist of input convolutional layers adapted for the Atlatl game environment (adaptation is based on hexagonal convolutional layers [4]) and then traditional dense sequential layers. Network construction for the Bayesian and non-Bayesian networks are shown in Chapter 5 and in table 2.1. The Bayesian models are based on the works of [5] and are expanded on in chapter 5. Because Atlatl uses a hexagonal grid system, traditional convolutional layers do not perform as well as convolutions designed for hexagonal grids. For example, a square only has four available directions when moving one unit distance. A hexagon has six. To compensate for this, hexagonal convolutions must be shifted on the y-axis in an alternating pattern. Figure 2.2 illustrates how the geometry is generally manipulated within the convolutions. Traditional post-convolution dense layers leverage the power of deep ML.

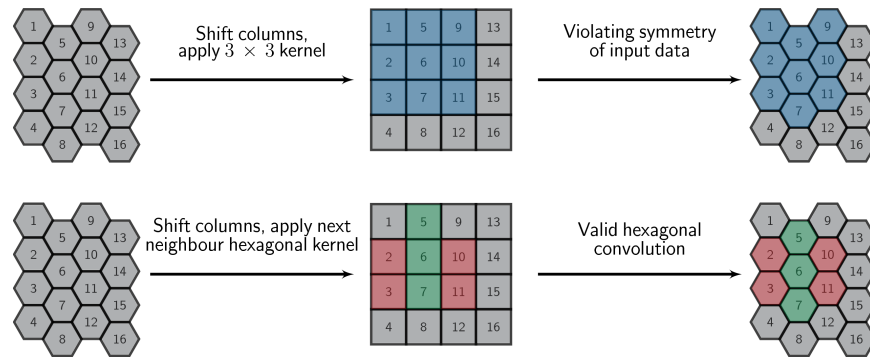


Figure 2.2. Illustration of hexagonal convolutions. Source: [4]

This network architecture was selected for two primary reasons. One, through the above-referenced work, they are generally known to work in the Atlatl environment, and two, the linear layers could be easily adjusted to a Bayesian implementation.

Table 2.1. The implemented non-Bayesian network

Layer (type)	Output Shape
Conv2d-1	[-1, 64, 5, 5]
ReLU-2	[-1, 64, 5, 5]
HexBlock-3	[-1, 64, 5, 5]
Conv2d-4	[-1, 64, 5, 5]
ReLU-5	[-1, 64, 5, 5]
HexBlock-6	[-1, 64, 5, 5]
Flatten-7	[-1, 1600]
Linear-8	[-1, 1600]
ReLU-9	[-1, 1600]
SlimFC-10	[-1, 1600]
Linear-11	[-1, 512]
ReLU-12	[-1, 512]
SlimFC-13	[-1, 512]
Linear-14	[-1, 512]
ReLU-15	[-1, 512]
SlimFC-16	[-1, 512]
Linear-17	[-1, 7]
SlimFC-18	[-1, 7]

2.3 Algorithms

This paper utilizes a Proximal Policy Optimization (PPO) [6] and Importance Weighted Actor-Learner Architecture (IMPALA) [7] to train the network to maximize a reward received from the training environment. Proximal Policy Optimization (PPO) is a type of reinforcement learning algorithm that seeks to improve the efficiency and stability of policy gradient methods. It optimizes a surrogate objective function that limits the extent of policy updates, reducing the risk of harmful, large updates while still facilitating continuous learning and improvement. By clipping the policy ratio, PPO keeps the updates close to

the current policy, which makes it more sample-efficient and less prone to performance collapses than standard policy gradient methods.

IMPALA is another reinforcement learning algorithm designed to scale up distributed reinforcement learning by sharing experiences across many actors who independently interact with copies of the environment. Each actor sends its trajectory to a centralized learner, which computes policy and value function updates.

IMPALA differs from PPO in several critical ways²;

1. Scalability and parallelism: IMPALA is designed to handle large-scale distributed training with many parallel environments and learners. It decouples the acting and learning processes, allowing for asynchronous updates and increased sample efficiency. This architecture helps scale the training process to thousands of actors and learners while maintaining stability.
2. Policy Optimization: PPO is an on-policy algorithm, meaning it optimizes the policy using the same data generated by the current policy. It introduces a “clipping” technique to prevent the updated policy from deviating too much from the old policy, ensuring stable and reliable policy updates. IMPALA, however, is an off-policy algorithm. It uses a vital sampling technique called V-trace to correct for the difference between the behavior policy (the policy that generated the data) and the target policy (the policy being learned). This allows IMPALA to use off-policy data, increasing sample efficiency.
3. Complexity: IMPALA introduces a more complex architecture with separating actors and learners and the V-trace correction, which requires additional computation and bookkeeping.

2.4 Thesis Reinforcement Learning Framework

2.4.1 Parallelization Reinforcement Learning

Parallelization is a technique used to speed up the training of RL algorithms by performing multiple computations simultaneously. There are several ways to parallelize RL training,

²Summary of [7] and [8] generated with ChatGPT, 18 May 2023, and rephrased for content and style.

including:

1. Data parallelism: In this approach, multiple copies of the same model are run on different instances of the target simulation. Each copy of the model agent independently updates a buffer for a designated “learner” node. This technique can be helpful when gathering a large amount of training experience concurrently [9], [10], [7].
2. Asynchronous methods: In this approach, multiple instances of the agent are run in parallel, each instance exploring the environment and updating its parameters independently. The updates from the different instances are then combined in some way, such as by taking the average, to update the parameters of the overall agent. This technique can be helpful when the environment is complex, and exploration is time-consuming [9], [10], [7].

Overall, parallelization can significantly speed up the training of reinforcement learning algorithms, making it possible to train models faster.

2.4.2 Ray Implementation

The open source frameworks used in this thesis implementation are openAI Gym [11]. The parallelization framework for reinforcement learning is called “Ray” [12].

The framework “Ray” is divided into several modules. The most essential “Ray Core” “provides a small number of core primitives (i.e., tasks, actors, objects) for building and scaling distributed applications” [12]. Ray Core, in effect, allows for paralleling Python functions and programs. The next module, “Ray RLLIB,” provides specific APIs and RL algorithms³. Most significantly, Ray RLLIB provides interfacing functions that allow interoperability with the OpenAI Gym standard [11] utilized by the Atlatl environment. Other models, such as “Ray Tune,” allow for hyper-parameter orchestration and training optimization, and “Ray Serve” provide components for serving trained models. This thesis leverages Ray Tune in training and Ray Serve in model “bottling” and deployment.

³Both the PPO and IMPALA algorithms used were imported from Ray RLLIB

CHAPTER 3:

Parallelization of Reinforcement Learning Applications

3.1 Problem Framing

Training AI models used to take a significant time and required exceptional computing power. Today with improvements in available computation capacity (cloud computation and academic supercomputer clusters) and the democratization of open source techniques (Pytorch and Tensorflow) the barrier for practically developing AI models is significantly lower. This chapter specifically looks to show how parallelization can be leveraged to improve training times without appreciably lowering average performance.

3.2 Design of Experiment

3.2.1 Overview

To test the speed-up achievable with a parallelization framework. I used a neural net based on [1]. The network consists of two sets of convolutional layers, each applying hexagonal convolutions; the first receives input from the environment's 14x5x5 feature state matrix of Atlatl and is convolved into 64 5x5 layers. Another 64x5x5 convolutional layer is followed by a flattening layer, then a series of linear layers decreasing until an ultimate size seven linear layer terminates the network, Figure 2.1 shows the tested network. With this neural net, 56 training runs were conducted. Each run was performed on City-Inf-5 for a total of 5 million time steps. Runs were divided into one of four categories;

1. A PPO trained model with CPUs only.
2. A PPO trained model with CPUs + GPU.
3. A IMPALA trained model with CPUs only.
4. A IMPALA trained model with CPUs + GPU.

3.2.2 Experiment Organization

Each category was assigned between 1 and 90 CPUs to operate as workers. Every run allocated eight dedicated CPUs for the “Head” node (node at which actual training occurs, i.e. traditional back-propagation and optimization) occur. Each “run”, for example, PPO CPU-only with 10 workers, had a single independent data collection period. Table 3.1 shows the specific organization of runs. Note, the CPU + GPU runs end at 40 worker CPUs due to Hamming supercomputer node constraints. Each section will include a table of time results in hours, and a combined results table is displayed in table 3.6.

Table 3.1. Table of PPO and IMPALA Runs per number of assigned CPUs

CPU																			
PPO	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
IMPALA	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90

CPU + GPU							
PPO	1	5	10	15	20	25	30
IMPALA	1	5	10	15	20	25	30

3.2.3 Reward Vs Score, and Optimality

Atlant as a simulation “grades” performance in a given scenario based on a points score. Points are typically rewarded based on destroying enemy units and occupying city tiles. Internally to Atlant, at the ML level, training is based on “reward”. Reward and score are similar, they are often correlated but they are not the same. The reward is distinct in that it is directly coupled to actions the ML network takes at a given state, the score is a rolling value across the course of a game. All figures in this thesis that display “episode reward mean” show the mean reward value over some number of games. Because of the nature of City-Inf-5, some scenarios are nearly always guaranteed to have low or high scores and correlated reward values. Taking a mean of several episodes effectively smooths the reward value so that trends may be more easily observed. Based on this thesis, previous work, and anecdotal experience, the “optimal” mean reward score is around ~110-120 for this scenario. To date, there is no way to prove if this is truly “optimal”.

Table 3.2. Table of PPO CPU parallelization results

PPO CPU	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Time (h)	60.5	55.4	55.4	97.2	35.8	55.6	57.4	53.1	95.4	40.1	37.3	38.4	53.4	54.6	53.5	54.4	54.3	53.4	53.0

3.3 PPO Results

3.3.1 PPO CPU Parallelization

PPO as an algorithm, is not optimized for parallelization. Figure 3.1 shows the results from the CPU-only parallelization runs. Adding additional worker nodes, to improve the rate of experience gain, did not appreciably affect the training run time. This is mostly because PPO, as an on-policy algorithm, has no mechanism for distributing experience gain. Additionally, there is no architectural or algorithmic reason why PPO CPU only, with 15 and 40 required over 90 hours each. I accredit this to poor hyper-parameter initialization.

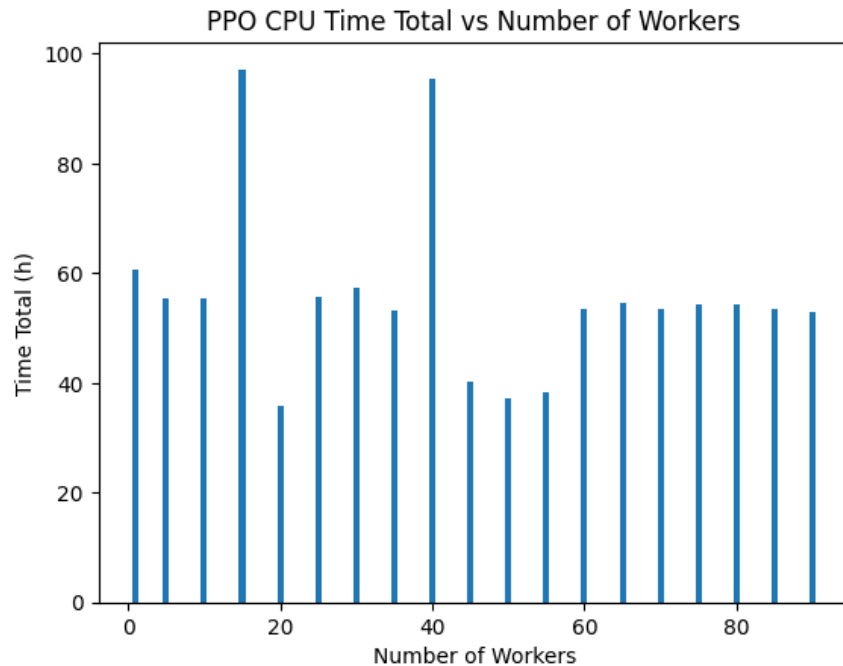


Figure 3.1. PPO vs CPU Training Time

3.3.2 PPO CPU + GPU parallelization

When supplied with a GPU, the PPO algorithm trained in a significantly lower time. CPU-only results averaged ~50 hours, including a GPU and any number of worker CPUs beyond five lowered training time to ~5.5 hours. This is expected, the presence of a GPU's parallel cores provides a significant improvement to the speed at which the underlying Torch library can process the forward and back propagations while learning. Figure 3.2 illustrates the times.

Table 3.3. PPO with CPU + GPU

PPO GPU	1	5	10	15	20	25	30
Time (h)	10.8	6.6	5.7	5.5	5.5	5.4	5.4

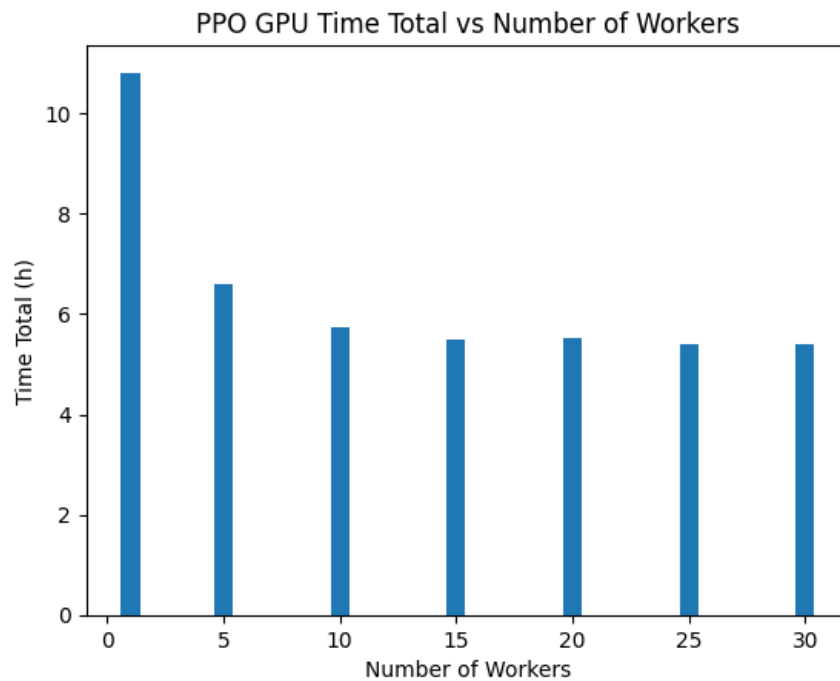


Figure 3.2. PPO CPU + GPU Results (h)

3.3.3 PPO Summary Comparison

Figure 3.3 shows the PPO CPU and PPO CPU + GPU results overlaid. These results, displayed in combined Figure 3.3 provide the baseline for comparing to the selected parallel algorithm IMPALA, examined in the next section. Additionally, Figure 3.4 shows the highest episode reward means achieved out of the PPO CPU-only, and PPO CPU + GPU runs. This illustrates that for each set of runs, the ML models that achieved the highest scores per experiment set, still achieved roughly equivalent rewards means regardless of time, ~35 hours to ~12 hours.

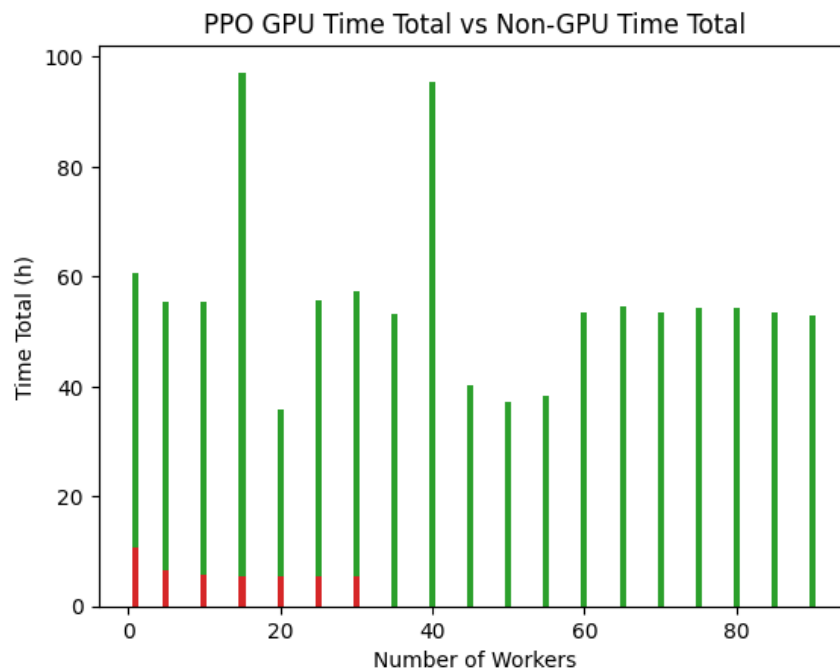


Figure 3.3. PPO GPU overlaid over CPU only training time

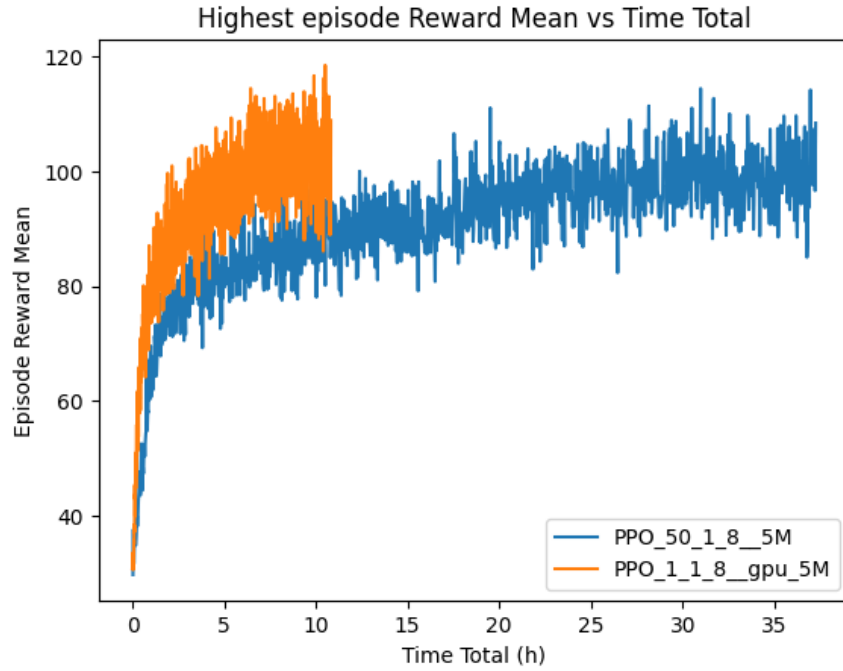


Figure 3.4. PPO Best CPU only vs Best CPU + GPU

3.4 IMPALA Results

3.4.1 IMPALA CPU Parallelization

The results from the IMPALA CPU-only experiment are displayed in Figure 3.5 and Table 3.4. The following observations are of note:

- When run with one worker, the training time was significantly worse than all other runs.
- For all runs with greater than 1 worker CPU the algorithm did not see any significant speed-up relative to itself. Taken together, I assess these two observations to show that beyond 5 worker CPUs, the experience gathering rate of the workers is greater than the rate at which the head node (equipped with 8 training CPUs) can perform the learning portion of RL learning and distribute updated policies.
- The approximate training time for all non-one CPU workers was ~1.5 hours. This outperformed both PPO CPU (with times ~50 hours) and PPO CPU + GPU (with

times ~5 hours) by factors of over 30x and 3x, respectively. IMPALA training with 13 CPUs (8 for the head node and 5 workers) significantly outperformed PPO with 98 CPUs and 38 CPUs + 1 GPU.

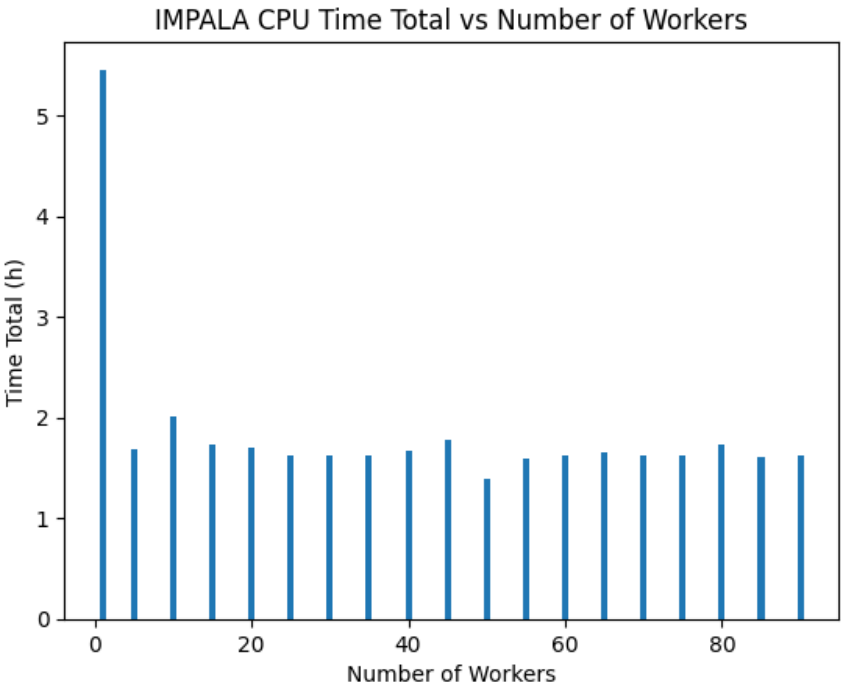


Figure 3.5. IMPALA vs CPU training time

Table 3.4. IMPALA CPU Results

IMPALA CPU	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
Time (h)	5.5	1.7	2.0	1.7	1.7	1.6	1.6	1.6	1.7	1.8	1.4	1.6	1.6	1.7	1.6	1.6	1.7	1.6	1.6

3.4.2 IMPALA CPU + GPU parallelization

Table 3.5 and Figure 3.6 show the results of the IMPALA with CPU + GPU experiment. The following observations are of note:

- There is a clear pattern of increasing performance with added CPU workers and a clear expression of diminishing returns with each additional CPU worker added.
- There is a significant performance jump from one worker to 5 or more workers.
- The overall shape of the results is distinctive from that of the CPU-only experiment shown in Figure 3.5.

Taken together, I assess that with a GPU available for training, initially with few workers, the GPU can complete training iterations faster than the workers can fill a new experience buffer. As more workers are added, the time required to fill the experience buffer decreases linearly, and each subsequent worker lowers the time needed by a decreasing proportional amount. This assessment accounts for the behavior of diminishing returns and rapid lowering of training time. Additionally, with many workers ~20 to 30, IMPALA CPU + GPU significantly outperforms all previous experiments in the time domain.

Table 3.5. IMPALA CPU + GPU Results (h)

IMPALA CPU + GPU	1	5	10	15	20	25	30
Time (h)	5.9	1.7	1.0	0.7	0.6	0.5	0.5

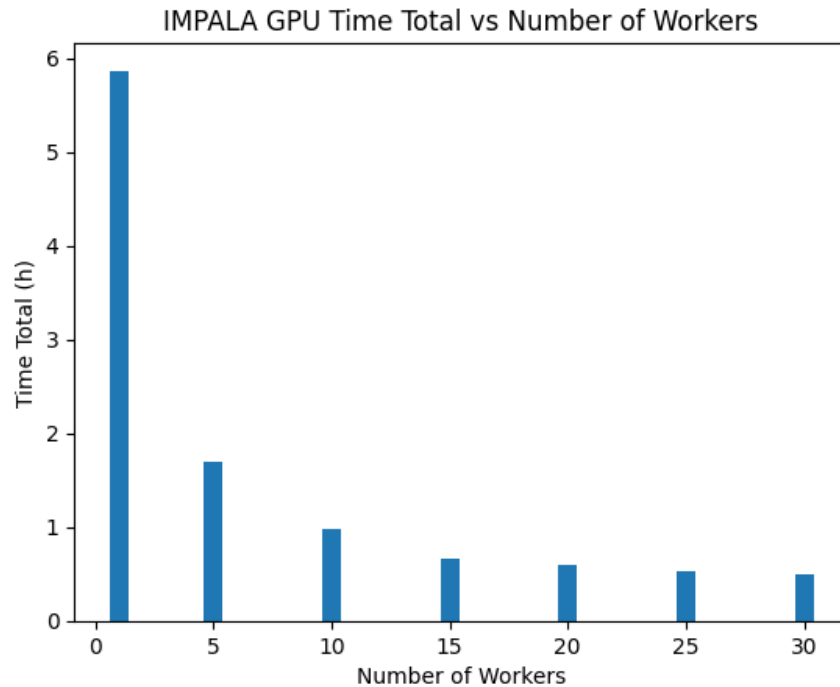


Figure 3.6. PPO w/GPU Training Time

3.4.3 Summary Comparison

As with PPO, IMPALA CPU + GPU outperformed IMPALA CPU only. The addition of a GPU unsurprisingly accelerated training times (a GPU’s parallel cores provide a significant improvement to the speed at which the underlying Torch library can process the forward and back propagation while learning). Figure 3.7 shows both IMPALA experiments overlaid, and Figure 3.8 illustrates the highest scores across both experiments compared to verify training time gains still achieved comparable scores.

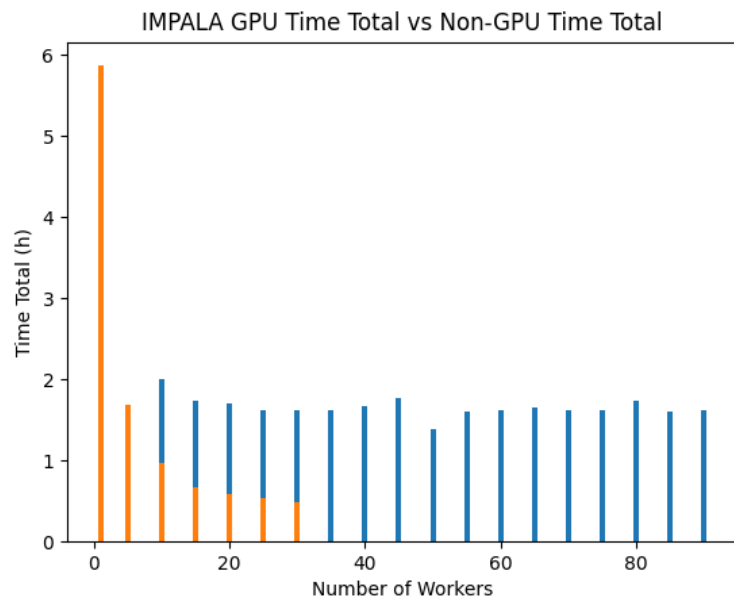


Figure 3.7. IMPALA CPU and CPU + GPU overlaid

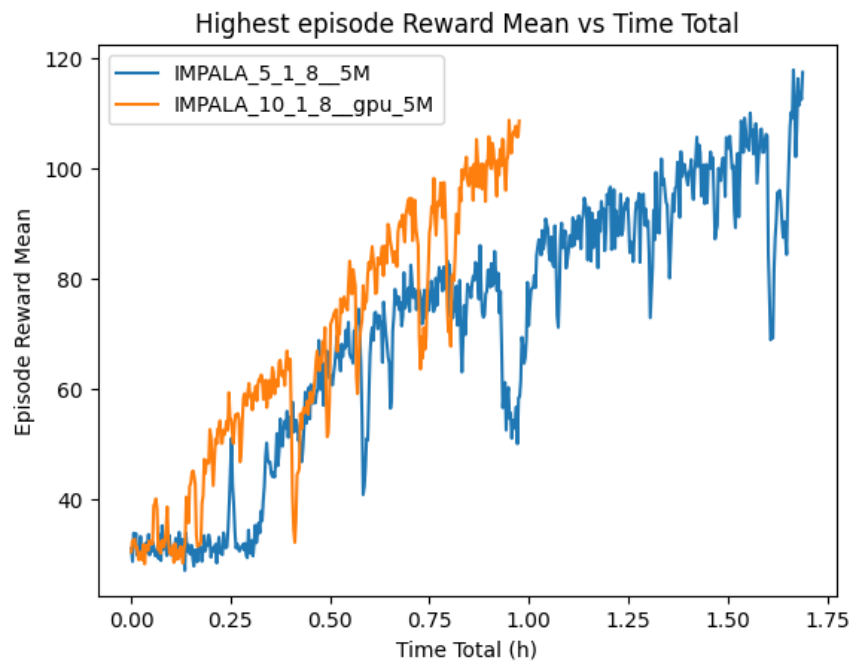


Figure 3.8. IMPALA Best CPU only vs Best CPU + GPU

3.5 Overall PPO vs IMPALA Comparison

With the fasted training time of ~.5 hours with 30 workers, IMPALA CPU + GPU significantly outperforms all other experiments detailed above by the following amounts (times were calculated by taking all run times in an experiment, removing the one worker CPU run time, and finding the average):

- PPO CPU only: ~80x speed improvement.
- PPO CPU + GPU: ~8x speed improvement.
- IMPALA CPU only: ~2.5 speed improvement.

Table 3.6 and Figure 3.9 show all experiment runs displayed together. Additionally, Figure 3.10 illustrates how across all four experiments, IMPALA CPU + GPU performs comparably in mean reward performance in significantly less time.

Table 3.6. All experiment timing results

	1	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
PPO CPU	60.5	55.4	55.5	97.2	35.8	55.6	57.4	53.1	95.4	40.1	37.3	38.4	53.4	54.6	53.5	54.4	54.3	53.4	53.0
PPO CPU+GPU	10.8	6.6	5.7	5.5	5.5	5.4	5.4												
IMPALA CPU	5.5	1.7	2.0	1.7	1.7	1.6	1.6	1.6	1.7	1.8	1.4	1.6	1.6	1.7	1.6	1.6	1.7	1.6	1.6
IMPALA CPU+GPU	5.9	1.7	1.0	0.7	0.6	0.5	0.5												



Figure 3.9. All time results by experiment

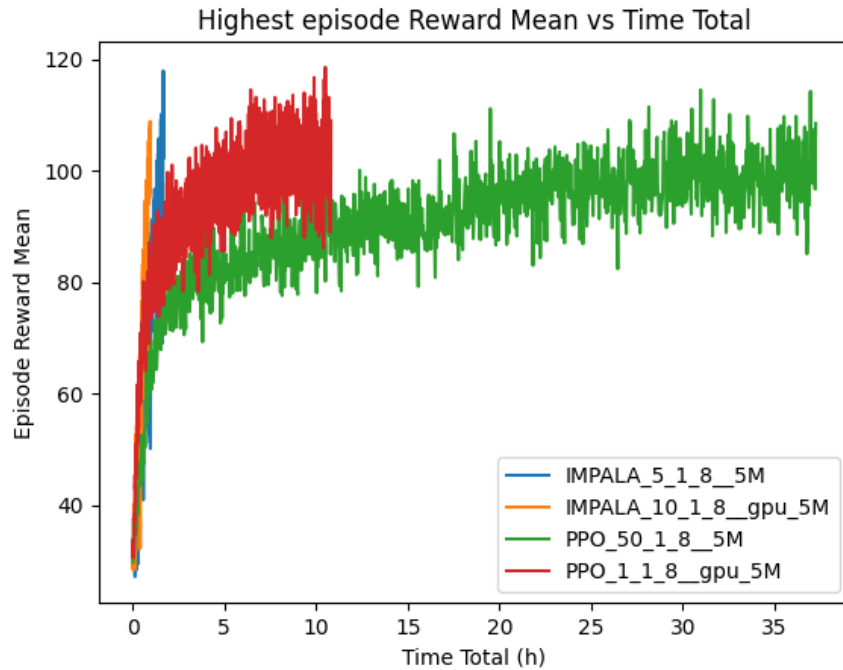


Figure 3.10. PPO and IMPALA Highest episode reward Mean vs Time Total

3.6 Additional Parallelization Conclusions

From the experiments above, I conclude the following:

- With a non-parallelized algorithm, such as PPO, there is no practical gain from operating with additional workers in parallel.
- With a modest amount of worker CPUs, a parallelized algorithm can perform equally in mean reward score in significantly less time with only CPUs than a GPU-enabled serial ML algorithm.
- With time performance gains between one and two orders of magnitude, there is a significant case for continuing to invest in parallel processing frameworks for RL training.
- As seen in Figure 3.8 IMPALA training with and without a GPU, with relatively few workers (~5 and 10), still achieves a mean reward score between 100 and 120. This is consistent with the other 5 million training step runs. The significant gain in time performance does not result in an appreciable loss in the mean reward score.

CHAPTER 4:

Azure Cloud Integration

4.1 Problem Framing

As computing power and ML capabilities increase, the speed and diversity of emerging simulations and tools will only increase. The ability to develop a wargame, allow human subject matter experts to build custom models, and deploy these models for consumption is critical to training and preparing future warfighters. Figure 4.1 illustrates how a vendor-developed wargame (or simulation) may be required to operate across several cloud environments and remote local deployments, Red Cells may need to develop ML models at higher classification levels than they are consumed, and models may be of a complexity that it becomes prohibitive to deploy them locally. This chapter demonstrates the development of a proof-of-concept solution to these problems. The data path for this proof-of-concept deployment is shown in Figure 4.2.

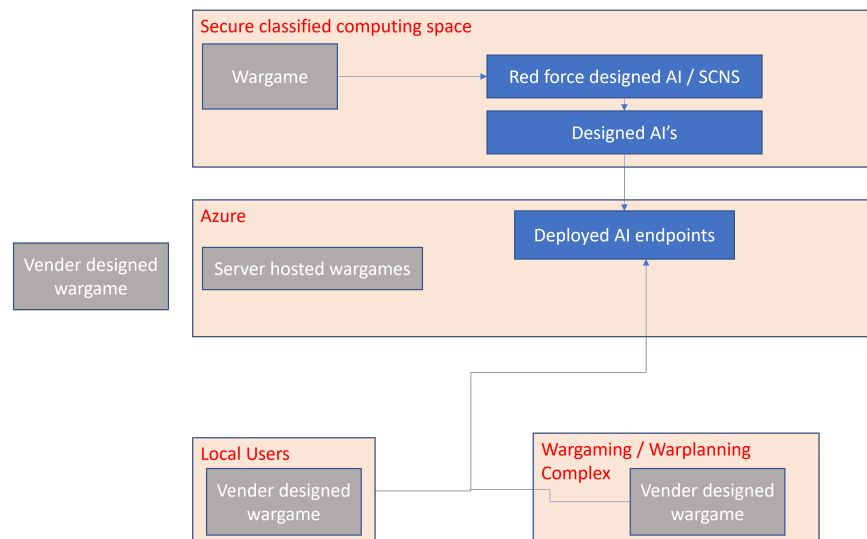


Figure 4.1. Future relationships between wargame developers and users across cloud and local instances

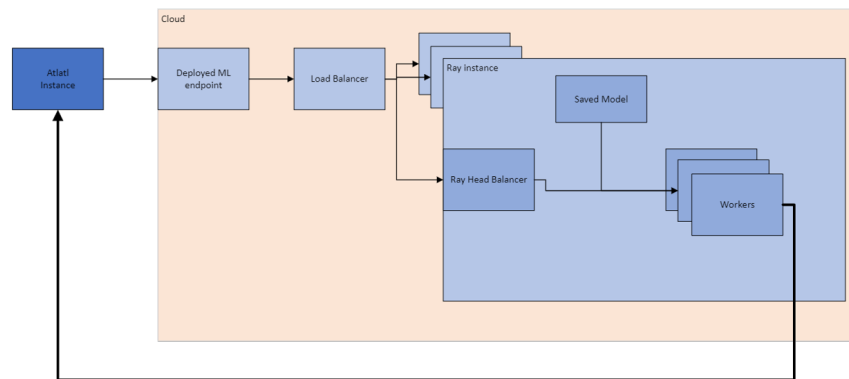


Figure 4.2. Functioning networking and a data path for deployed inference containers

4.2 Solution Overview

The solution proffered in section 4.1 is accomplished with the following steps:

1. Take a trained ML model developed with the parallelization framework used in section 3 and “bottle” it for deployment.
2. Build a Docker container built on an Azure Machine Learning base image, customized for the custom Ray and Atlatl framework.
3. Instance a compute cluster and deploy the “bottled” ML model with an Internet Protocol invocable call.
4. Show a functioning web API call to the deployed model, sending a game’s feature state information and receiving the trained models’ action.

4.3 Environment Development

4.3.1 Docker Build Parameters

Building the environment for remote deployment uses the Docker framework [13]. Figure 4.3 represents the relevant portions of the Docker code. The following sections are of note;

- Lines 1-19 allow container orchestration within Azure services and extended from Azure curated ML images.

- Lines 20-24 install the Python prerequisites for running trained Atlatl ML models.
- Lines 25-28 allow for the internal hosting of servers to handle calls between internet deployment points and model inference capabilities.

```

1 FROM mcr.microsoft.com/azureml/inference-base-1804:20230109.v1
2 RUN conda create -n minimal python=3.7 pip=22.1.2
3 ENV AZUREML_CONDA_ENVIRONMENT_PATH /opt/miniconda/envs/minimal
4 # Prepend path to AzureML conda environment
5 ENV PATH $AZUREML_CONDA_ENVIRONMENT_PATH/bin:$PATH
6 RUN pip install 'azureml-inference-server-http~=0.7.0' \
7         'ipykernel~=6.0' \
8         'azureml-core==1.48.0' \
9         'azureml-dataset-runtime==1.48.0' \
10        'azureml-defaults==1.48.0' \
11        'azure-ml==0.0.1' \
12        'azure-ml-component==0.9.15.post2' \
13        'azureml-mlflow==1.48.0' \
14        'azureml-contrib-services==1.48.0' \
15        'azureml-contrib-services==1.48.0' \
16        'torch-tb-profiler~=0.4.0' \
17        'py-spy==0.3.12' \
18        'debugpy~=1.6.3' \
19        'ray[rllib]==2.2.0' \
20        'joblib' \
21        'ray[serve]' \
22        'hexagdly' \
23        'gym==0.23.1' \
24        'requests' \
25        'torch' \
26        'starlette' \
27        'numpy'
28
29 ENV LD_LIBRARY_PATH
30 $AZUREML_CONDA_ENVIRONMENT_PATH/lib:$LD_LIBRARY_PATH
31 CMD [ "runsvdir", "/var/runit" ]

```

Figure 4.3. Docker code block for building deployment Docker container

4.3.2 Docker deployment options

Leveraging the Docker ecosystem allows deploying the trained ML model to various hosts. Illustrated in Figure 4.4, the build image may be deployed to either local hosts or remote hosts, provisioned on dedicated local hardware or scalable, dedicated cloud inference clusters. This flexibility in deployment targets address the envisioned configuration illustrated in Figure 4.1

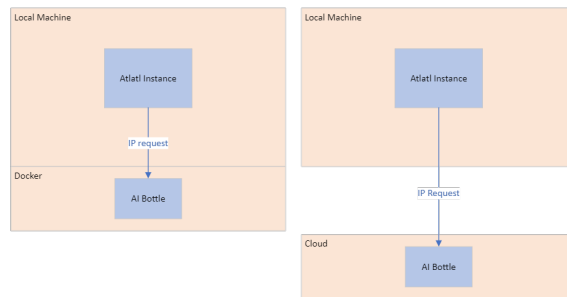


Figure 4.4. Flexible image deployment options

4.4 Initialization and Deployment

4.4.1 Initialization

After the image created in section 4.3.1 loads, an executable Python file is supplied that contains all of the script needed to initialize and run follow-on inference requests, key components of this script include:

1. Imports of required libraries, Ray, RLLIB, Ray Serve, torch, gym, starlette, requests, etc.
2. The code required to copy the implementation of a trained ML model.
3. A “init()” function that creates a request server and instantiates and loads the saved ML model.
4. A “run()” function handles receiving raw data, performing an inference, and returning data to the running server.

4.4.2 Initialization Function

Initialization with “init()” leverages the Ray Serve module (introduced in section 2.4.2 to effectively recreate the structure⁴ of a trained model and load the learned weights into it. Once loaded, “init()” uses Ray Serve to start a Starlette server to handle the request/response processing. These preparations correspond to circle points one and two in Figure 4.5.

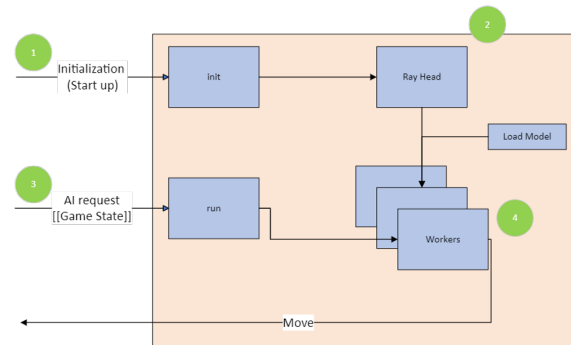


Figure 4.5. Initialization and serving data paths within deployed inference containers

4.4.3 Starlette Server

Starlette [14] is an Asynchronous Server Gateway Interface (ASGI) web framework that can be used to build high-performance asynchronous web services and APIs in Python. A Starlette server is a web server that uses the Starlette framework to handle HTTP requests and responses.

Starlette provides a simple and lightweight API that makes it easy to define routes, handle requests, and return responses. It also supports asynchronous programming, allowing developers to write efficient and scalable code that can handle a large number of concurrent requests.

One of the key features of Starlette is its support for async/await syntax, which allows developers to write non-blocking code that can handle multiple requests simultaneously without blocking the event loop. This can lead to improved performance and scalability,

⁴“Structure” meaning the shape of the model’s layers, a shell that fits the trained model’s weights to be loaded.

especially for high-traffic web applications.⁵

4.4.4 Deployment

Once deployed, the model is given an Internet Protocol (IP) endpoint Uniform Resource Locator (URL). A requesting process may then submit a IP packet with the body consisting of data = [[The current Atlatl state feature space, a 14x5x5 matrix]]. Once a request is received, “run()” extracts the features using JavaScript Object Notation (JSON) and transforms it into a numpy array. This array is passed to an inferencing function, and the results are returned as a move choice value of 0 through 6 (inclusive), corresponding to the trained model’s action space in Atlatl. The “run()” and prompt/response handling corresponds to circle points three and four in Figure 4.5.

4.4.5 Potential Configurations

I deployed my model to a relatively small, single-instance CPU-only cluster in this proof-of-concept deployment. Figure A.1 shows Azure metric logs from that deployment. Even with a small inference serve, the deployed endpoint was capable of handling hundreds (consistently between 100-200 requests per minute), at ~24 ms of latency. The inclusion of more inference nodes or of GPU-enabled nodes would likely lower latency and increase the ability of the endpoint to handle more requests. The use of Ray Serve and the internal Starlette server allows for variation in the size of the model’s deployment. This configuration allowed for prompt/response to be handled serially. If deployed within a larger pool of inference nodes, where each node was of multiple CPUs or GPUs, this process could be scaled up and automatically load balance across nodes. The degree of parallelization and associated cost is offered as future work in this area.

4.5 Benefits of cloud deployment

Currently, deployment of RL models to the cloud offers little to no utility. In the future, if models built with RL techniques need to be trained and consumed by different organizations⁶, the need to decouple locally trained models from local environments, and deployment

⁵Summary of [14] generated with ChatGPT, 18 May 2023, and rephrased for content and style.

⁶There may be dedicated Red and Blue Cells modeling behaviors and another Cell for experimentation and evaluation

of model the cloud will be a necessary part of the needed DevOps pipeline. Ensuring the training framework is already integrated with a deployment and serving framework will be required to reduce overall cost and production times.

CHAPTER 5:

Bayesian Implementation

5.1 Problem Framing

AI and ML algorithms have made tremendous strides in recent years. However, most models have been deterministic, meaning they produce the same output for a given input every time. While deterministic models can be accurate, they cannot provide a measure of "certainty" or the level of confidence that the model has in its predictions. This shortcoming is particularly relevant in the field of wargaming as it moves towards human-machine teaming, where the ability to communicate and interpret the level of confidence of an AI for a given game state is critical.

One approach to address this issue is implementing Bayesian methods in the ML model. Bayesian methods can be instrumental in providing a measure of certainty during gameplay or where probabilistic predictions are required. By incorporating Bayesian methods in AI systems, we can create models that can estimate the degree of certainty in their predictions and establish meaningful ways to communicate those confidence levels to humans. Such approaches may help move the field closer to dependable and transparent intelligent systems that can be better integrated with human decision-making.

5.2 Applying Bayesian Methods to Neural Networks

Kullback-Leibler (KL) divergence, known as relative entropy, measures the difference between two probability distributions. It quantifies the amount of information lost when a probability distribution is used to approximate another distribution.

$$\mathbf{D}_{KL}(P|Q) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2} \quad (5.1)$$

Equation 5.1 [15] illustrates KL divergence between probability distributions P and Q, for the case of KL divergence between two Gaussian distributions where σ_1 and σ_2 are the

standard deviations of distribution p and q , μ_1 and μ_2 are the means of distribution p and q , equation 5.1 may be used to calculate the divergence.

In a neural network, the KL loss term constrains the learned probability distribution to be close to some target distribution, often a simple prior distribution. The loss function can be used to minimize the difference between the two distributions to sequentially improve a model while maintaining weight values from previously learned experience.⁷

In essence, a ML network, built with Bayesian techniques, should be able to show a ML model's uncertainty for a given game state. Figure 5.1 shows how this network could theoretically look. Atop is a traditional deterministic model. Given some game state, the model will always choose the same move. Below that is a model with Bayesian layers. Weight values have been replaced with learned means and deviations. Given some game state, there is a probability distribution of the move selection the model will take. If the model has seen constant rewards from some actions from a given state, then the distribution will be small, and few move choices will be present; conversely, if the model is uncertain, the distribution will be flatter, with more move choices appearing. Figure 5.2 illustrates notional probability distributions of high and low Bayesian models.

⁷Summary of [15] generated with ChatGPT, 18 May 2023, and rephrased for content and style

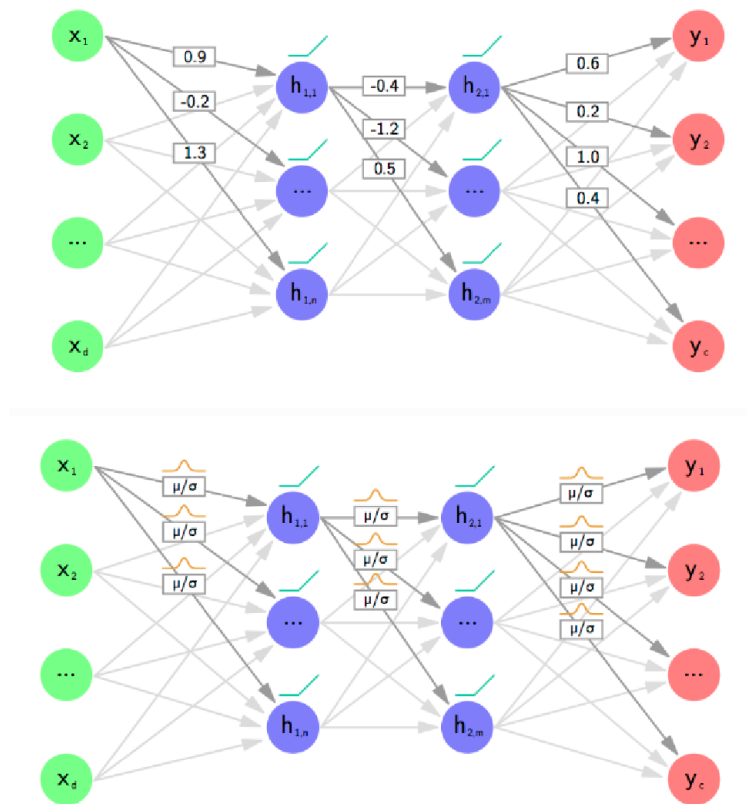


Figure 5.1. Illustration of a non-Bayesian (Deterministic) and a Bayesian (probabilistic) Neural Network. Source: [16]

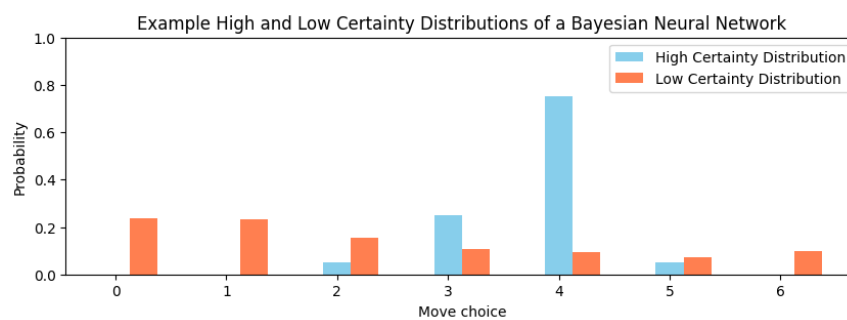


Figure 5.2. Illustration of a Bayesian network with high and low certainty.

5.2.1 Bayesian Variational Layers

For the Bayesian implementation, I have recreated the network using linear variational layers derived from [5] and [3] and modified the PPO loss function to include the sum of each layer's KL loss value. Table 5.1 has the implemented network.

To implement the addition of KL loss from the added Bayesian linear variational layers, I modified the PPO loss function with the following code block:

```
if !Bayesian_layer_present:
    kv_div = 0
    custom_loss = kv_div
else:
    for layer in model.torch_sub_model._convs:
        if isinstance(layer, linear_variational.LinearReparameterization):
            kl_div = layer.kl_loss()
            custom_loss += kl_div
    total_loss += custom_loss
```

This allows updating the total loss to include the KL loss from the linear variational layers.

5.2.2 Epistemic Uncertainty

Epistemic uncertainty in the context of RL learning measures how consistent a model's predictions are if repeatedly submitted to a trained model. A high certainty model will have a low distribution of move choice likelihoods (displayed in blue in Figure 5.2) whereas a low certainty will have move choice probabilities in a flatter, more disparate distribution (displayed in orange in Figure 5.2).

5.2.3 Calculation of epistemic uncertainty

Calculating a model's epistemic uncertainty for a given game state is done through several steps.

- A game state is passed through a trained model.
- A softmax activation function is applied to the outputted action distribution.

- Steps one and two are repeated until some desired number of results are acquired.
- The mean and standard deviation for each column are calculated.
- Epistemic uncertainty is then calculated as the sum of all variances from each column.

```

predictions = []
for i in range(50):
    temp_features = input_dict["obs"].float()
    temp_features = temp_features.permute(0, 1, 2, 3)
    prediction = self._convs(temp_features)
    predictions.append(torch.softmax(prediction, dim=1))
mean = torch.mean(torch.stack(predictions), dim=0)
variance = torch.mean(torch.square(torch.stack(predictions) - mean), dim=0)
Epistemic_uncertainty = torch.sum(variance).item()

```

5.3 Experiments

5.3.1 Neural networks

For this thesis, I implemented two networks: a convolutional network with no Bayesian layers and another with Bayesian layers. The networks were trained across several million time steps from the City-Inf-5 Atatl scenario. Mean and maximum game reward scores were recorded and plotted.

5.3.2 Results of Non-Bayesian convolutional network

Over approximately 1.2 million training episodes, the Non-Bayesian network achieves a max reward score of 450 and a mean reward score of about 90. These smoothed approximations are illustrated in Figures 5.3 and 5.4

Table 5.1. The implemented non-Bayesian network

Layer (type)	Output Shape
Conv2d-1	[-1, 64, 5, 5]
ReLU-2	[-1, 64, 5, 5]
HexBlock-3	[-1, 64, 5, 5]
Conv2d-4	[-1, 64, 5, 5]
ReLU-5	[-1, 64, 5, 5]
HexBlock-6	[-1, 64, 5, 5]
Flatten-7	[-1, 1600]
Linear-8	[-1, 1600]
ReLU-9	[-1, 1600]
SlimFC-10	[-1, 1600]
Linear-11	[-1, 512]
ReLU-12	[-1, 512]
SlimFC-13	[-1, 512]
Linear-14	[-1, 512]
ReLU-15	[-1, 512]
SlimFC-16	[-1, 512]
Linear-17	[-1, 7]
SlimFC-18	[-1, 7]

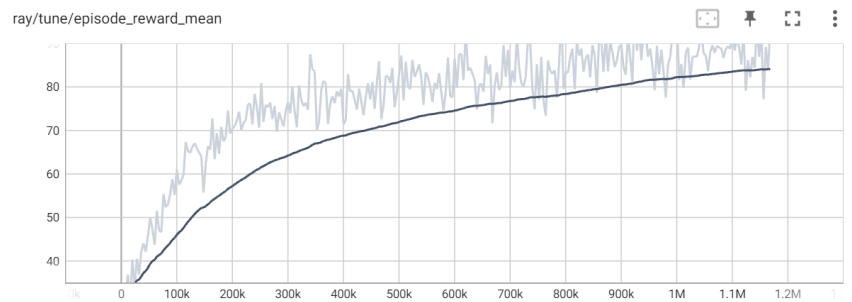


Figure 5.4. Graph of the mean reward during training. Non-Bayesian

Table 5.2. The implemented Bayesian network

Layer (type)	Output Shape
Conv2d-1	[-1, 64, 5, 5]
ReLU-2	[-1, 64, 5, 5]
HexBlock-3	[-1, 64, 5, 5]
Flatten-4	[-1, 1600]
LinearReparameterization-5	[-1, 1600]
LinearReparameterization-5	[-1, 800]
LinearReparameterization-6	[-1, 512]
LinearReparameterization-7	[-1, 512]
LinearReparameterization-8	[-1, 7]

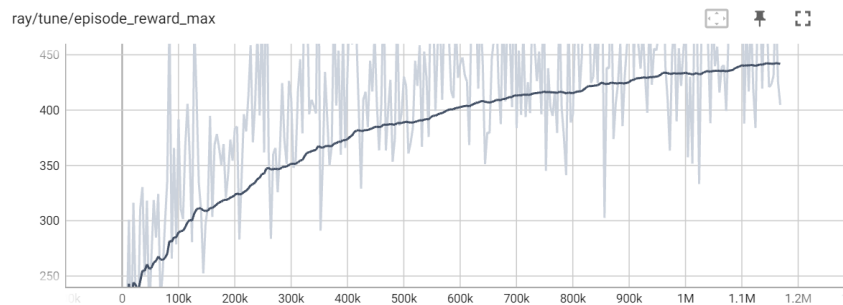


Figure 5.3. Graph of the maximum reward during training. Non-Bayesian

5.3.3 Results of Bayesian convolutional network

The training of the Bayesian work resulted in a mean reward score of approximately 55 and a maximum reward score of roughly 300; these values are illustrated in figures 5.6 and 5.5. These represent significant losses to the baseline Non-Bayesian networks but offer gains by being capable of providing a calculation of the model's epistemic uncertainty during run-time.

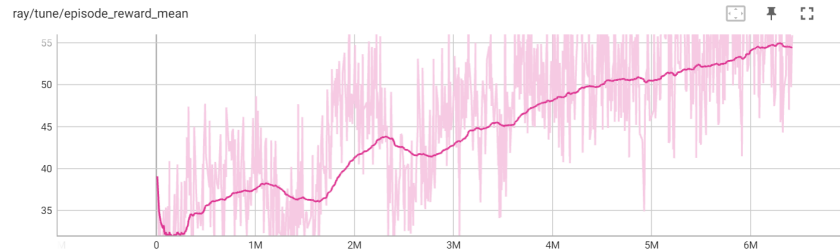


Figure 5.5. Graph of the mean reward during training, Bayesian Model

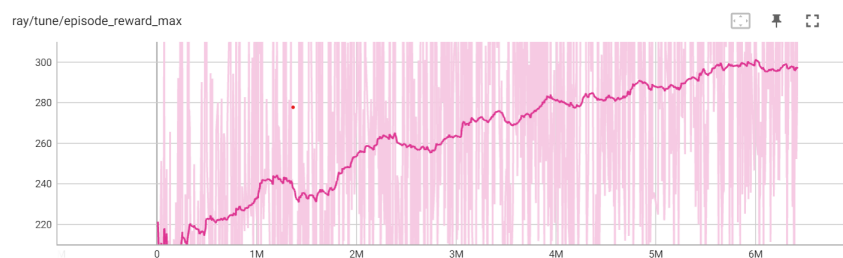


Figure 5.6. Graph of the maximum reward during training, Bayesian Model

5.3.4 Analysis of Epistemic Uncertainty

With the Bayesian variational layers, it is possible to plot the epistemic uncertainty of each move, in identical starting games, across different iterations of training. Figure 5.7 and Table 5.3 illustrate this. Early in training, the red 10 and 60 plots show a high degree of epistemic uncertainty across all moves in the game. I assess this is due to the model having trained very little, and every move choice is effectively a novel one. Much later in training, shown in blue plots 1850 and 1900, the epistemic uncertainty collapses to zero early on and only rises near the end. This translates to the AI being extremely confident during early moves as it directs units toward cities but is highly not confident as it nears units of opposing forces.

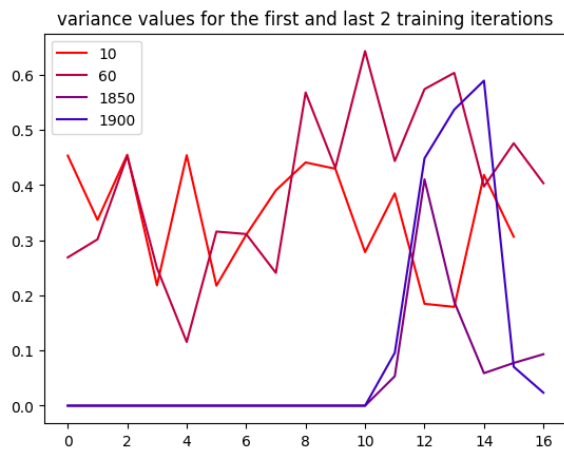


Figure 5.7. Change in epistemic uncertainty per move at four training iterations.

Table 5.3. Epistemic uncertainty per time step after 10, 60, 1850, and 1900 training iterations.

Training Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
10	0.45	0.34	0.45	0.22	0.45	0.22	0.31	0.39	0.44	0.43	0.28	0.38	0.18	0.18	0.42	0.31
60	0.27	0.3	0.45	0.25	0.12	0.32	0.31	0.24	0.57	0.43	0.64	0.44	0.57	0.6	0.4	0.48
1850	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.05	0.41	0.19	0.06	0.08	0.09
1900	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.45	0.54	0.59	0.07	0.02

5.4 Unexpected Scenarios

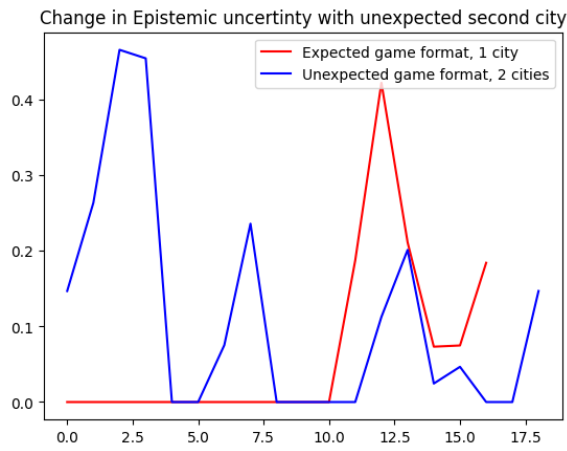


Figure 5.8. Graph of model uncertainty with additional city present.

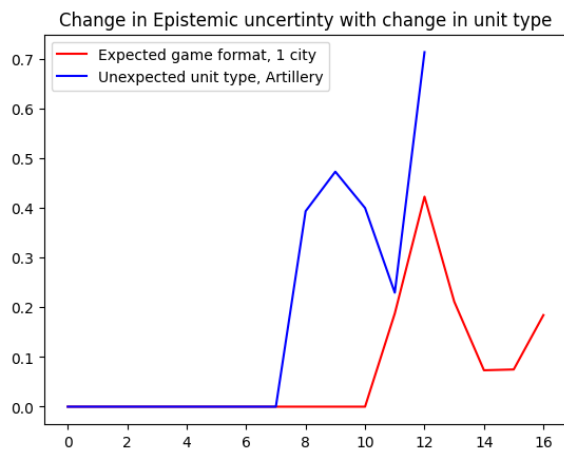


Figure 5.9. Graph of model uncertainty with new red force unit present.

Two additional scenarios were used to verify that the models were modeling uncertainty. One scenario modified cit-inf-5 to have a second city. Another scenario was altered so that a new, never-before-seen enemy unit type was placed in opposition to the blue forces; the unit was an “artillery” type, and this unit type could engage the blue forces sooner. Figure 5.8 clearly shows an increase in uncertainty sooner, this correlated to a blue unit needing to move toward one of the two cities or engage the opposing red forces, a decision the model

had never needed to make this early in a game before, the observed higher uncertainty confirms this. Likewise, Figure 5.9 shows a drastic increase in uncertainty sooner than the normal City-Inf-5; this corresponded to when the added artillery unit began firing on the blue forces, again something the model had never seen in training.

5.4.1 Conclusion

In conclusion, including Bayesian layers and techniques is entirely compatible with RL. Including variational layers adds a level of perception to the AI's model that could be extremely valuable in a human-machine teaming format. While there is still work to attain parity between performance scores, altered network structure and optimized hyper-parameters search may help.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:

Conclusion

6.1 Benefits of Parallelization

Parallelization, even without a GPU, can significantly reduce training times with little to no penalty to final model performance. Figure 6.1 shows the remarkable speed-ups possible. Additionally;

1. With a non-parallelized algorithm, such as PPO, there is no practical gain from operating with additional workers in parallel.
2. With a modest amount of worker CPUs, a parallelized algorithm can perform equally in mean reward score in significantly less time with only CPUs than a GPU-enabled serial ML algorithm.
3. With time performance gains between one and two orders of magnitude, there is a significant case for continuing to invest in parallel processing frameworks for RL training.

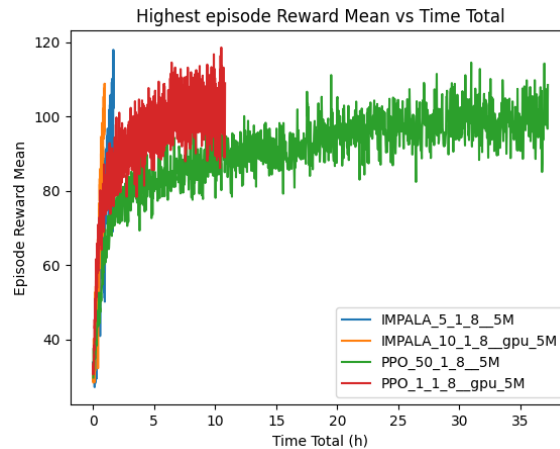


Figure 6.1. PPO and IMPALA Highest episode reward Mean vs Time Total

6.2 Practically Deploying Reinforcement Learning Trained Models

Currently, deployment of RL models to the cloud offers little to no utility. In the future, if models built with RL techniques need to be trained and consumed by different organizations⁸, the need to decouple locally trained models from local environments, and deployment of model the cloud will be a necessary part of the needed DevOps pipeline. Ensuring the training framework is already integrated with a deployment and serving framework will be required to reduce overall cost and production times.

6.3 Benefits of Bayesian Techniques in RL Models

Variational layers and, more abstractly, Bayesian techniques are entirely compatible with RL. A simple variational layer adds a level of perception to the AI's model that could be extremely valuable in a human-machine teaming format. While there is still work to attain parity between scores, altered network structure and optimized hyper-parameters search may help. Figure 6.2 pointedly illustrates how a human can receive information from a trained model on how confident it is on its decision and with further integration and research may enable further human-machine teaming and increase trust in a AI model's decisions.

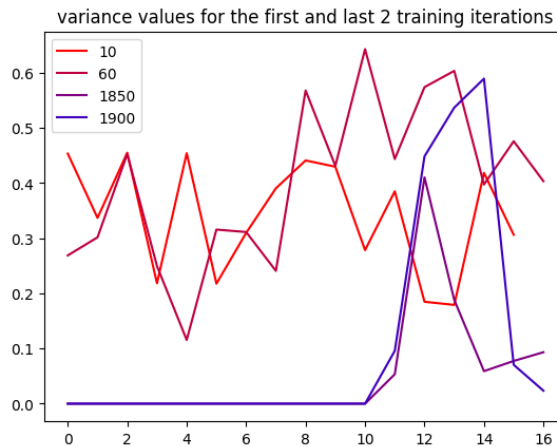


Figure 6.2. Change in epistemic uncertainty per move at four training iterations.

⁸There may be dedicated Red and Blue Cells modeling behaviors and another Cell for experimentation and evaluation

6.4 Recommendations for Future Work

6.4.1 Parallelization Future Work

This thesis focused on demonstrating the possible improvements when shifting from a Non-parallelized algorithm (PPO) to a parallelized framework and parallelized algorithm (RAY and IMPALA respectively). The following topics of study are:

- Effects of multiple GPUs on asynchronous by design algorithms like IMPALA
- Effects of parallelization on different neural architectures, i.e. test different convolutional and sequential layer combinations.
- Effects of changing the Atlatl feature space, with asynchronous workers and priority buffer queuing, there may be available speed-ups in increasing the size or complexity of the feature space.

6.4.2 Cloud Integration Future Work

Many possible refinements could be made to this work; reducing the size of the deployed image and identifying improved open-source standards and methods for deploying the “bottled” model are both possible areas. Analysis of “inference rate” to cost i.e. how many deployed clusters and what sizes are needed to ensure some target rate capacity or response latency, would also be of value.

6.4.3 Bayesian Methods Future Work

Implementation of Bayesian methods is an open field of research right now; the following are possible additional topics of study:

1. Analysis of what Bayesian layers most effectively implement into RL models.
2. Analysis of what training algorithms most effectively train Bayesian-based networks (i.e. PPO, DQN, IMPALA etc.)
3. Analysis of why average scores are lower, is it a function of the model’s network, the training algorithm used, or hyper-parameter tuning?
4. Create a wargame, with a AI model trained with and without Bayesian methods and assess how a human judgment is affected when given a AI decision with associated certainty.

6.5 Summary and Recommendations

In summary, this research contributes to advancing machine learning and wargaming simulation techniques by emphasizing the significance of parallelization, Demonstrating a RL model deployed to a cloud-based environment, and the utility of incorporating Bayesian methods for additional information to improve human-machine collaboration. To ensure these findings can be implemented in future research and practical applications, I make the following recommendations:

1. Simulations should be built to adhere to open-source standards; this work specifically adhered to the “OpenAI Gym” environment standards.
2. Training algorithms and models should be capable of and leverage parallelization by default.
3. Deployment of trained models should be part of the development process; trained models should, by default, be packaged and prepared for deployment to cloud environments. Additionally, deployed models should be incorporated into the training or verifying new models.
4. For human-machine-teaming-centered projects, basic Bayesian techniques should be foundational in models.

APPENDIX: REQUEST RATE AND LATENCY LOG METRICS

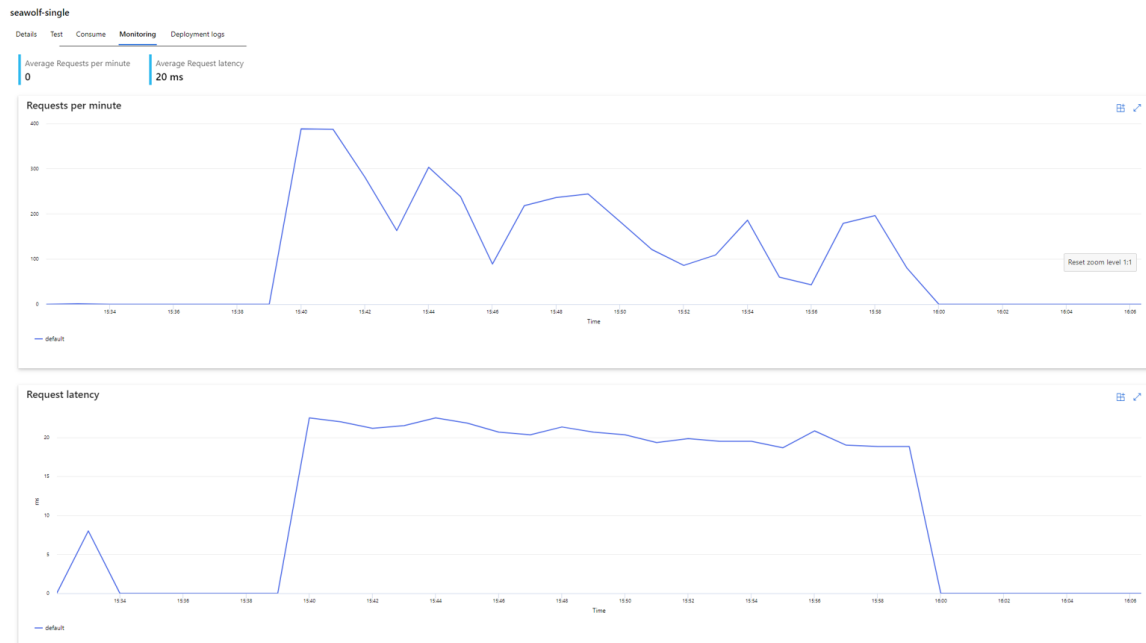


Figure A.1. Graph of deployed model latency and request rate

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

- [1] C. T. Cannon and S. Goericke, “Using convolution neural networks to develop robust combat behaviors through reinforcement learning,” 2021-06. Available: <https://calhoun.nps.edu/handle/10945/67681>
- [2] I. Allen, John T., “Enlisting ai in course of action analysis as applied to naval freedom of navigation operations,” 2022-09. Available: <https://calhoun.nps.edu/handle/10945/71041>
- [3] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural network,” in *Proceedings of the 32nd International Conference on Machine Learning* (Proceedings of Machine Learning Research), F. Bach and D. Blei, Eds. Lille, France: PMLR, 07–09 Jul 2015, vol. 37, pp. 1613–1622. Available: <https://proceedings.mlr.press/v37/blundell15.html>
- [4] C. Steppa and T. L. Holch, “Hexagdly—processing hexagonally sampled data with cnns in pytorch,” *SoftwareX*, vol. 9, pp. 193–198, 2019.
- [5] R. Krishnan, P. Esposito, and M. Subedar, *Bayesian-Torch: Bayesian neural network layers for uncertainty estimation*, <https://github.com/IntelLabs/bayesian-torch>, Jan. 2022. Available: <https://doi.org/10.5281/zenodo.5908307>
- [6] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” 2015. Available: <https://arxiv.org/abs/1502.05477>
- [7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, “IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures,” pp. 1407–1416, 10–15 Jul 2018. Available: <https://proceedings.mlr.press/v80/espeholt18a.html>
- [8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” New York, New York, USA, pp. 1928–1937, 20–22 Jun 2016. Available: <https://proceedings.mlr.press/v48/mniha16.html>
- [10] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, “Distributed distributional deterministic policy gradients,” *arXiv preprint arXiv:1804.08617*, 2018.

- [11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. Available: <http://arxiv.org/abs/1606.01540>
- [12] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan *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.
- [13] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.
- [14] Encode, "Encode/starlette: The little asgi framework that shines." "Accessed May. 22, 2023 [Online]". Available: <https://github.com/encode/starlette>
- [15] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, January 2006. Available: <https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/>
- [16] K. Shridhar, F. Laumann, and M. Liwicki, "A comprehensive guide to bayesian convolutional neural network with variational inference," *ArXiv*, vol. abs/1901.02731, 2019.

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California



DUDLEY KNOX LIBRARY

NAVAL POSTGRADUATE SCHOOL

WWW.NPS.EDU

WHERE SCIENCE MEETS THE ART OF WARFARE