Open University of Catalonia (UOC) Master's Degree in Data Science

MASTER'S THESIS

Area: Reinforcement Learning

Paradigms of Artificial Intelligence

A Compatibility Study between Feudal Reinforcement Learning and Federated Reinforcement Learning

---

Author: Marc Debés i Parés

Tutor: Luis Esteve Elfau

Professor: Ismael Benito Altamirano

Zurich, June 4, 2025

Credits/Copyright

FINAL PROJECT RECORD

Title of the project:	A Compatibility Study between Feudal Reinforcement Learning and Federated Reinforcement Learning
Author's name:	Marc Debés i Parés
Collaborating teacher's name:	Luis Esteve Elfau
PRA's name:	Ismael Benito Altamirano
Delivery date (mm/yyyy):	06/2025
Degree or program:	Master's Degree in Data Science
Final Project area:	Area 4
Language of the project:	English
Keywords	Reinforcement Learning, Feudal Reinforcement Learning, Federated Learning, Multiagent Systems, Privacy-Preserving AI

Abstract

This research project proposes a comprehensive compatibility study between two cutting-edge reinforcement learning paradigms: Feudal Reinforcement Learning and Federated Reinforcement Learning. In modern artificial intelligence, scalability and data privacy are paramount. Feudal Reinforcement Learning introduces a hierarchical structure in which a global Manager-Agent decomposes complex tasks by setting sub-goals, while local WorkerAgents execute these tasks by learning policies based on both their own observations and the manager's high-level directives. Federated Reinforcement Learning, in contrast, enables multiple distributed agents to collaboratively learn a shared model without centralized data collection, thereby preserving privacy and reducing communication overhead through techniques such as federated averaging and FedProx regularization.

The proposed study will rigorously analyze and benchmark these paradigms in multiagent environments simulated via the PettingZoo library. The evaluation focuses on metrics such as learning efficiency, convergence speed, and robustness in heterogeneous settings. This investigation is significant as it aims to develop a hybrid approach that leverages the strengths of both methods to address contemporary challenges in AI, particularly in scenarios where data sensitivity and scalability are critical. By synthesizing theoretical insights with empirical evidence, this work aspires to contribute to the advancement of privacy-preserving, scalable learning architectures for complex, real-world applications.

Keywords: Reinforcement Learning, Feudal Reinforcement Learning, Federated Learning, Multiagent Systems, Privacy-Preserving AI

Contents

List of Figures

List of Tables

Chapters

Chapter 1

Introduction

The rapid evolution of artificial intelligence has given rise to specialized methodologies for tackling increasingly complex problems. This proposal focuses on a compatibility study between Feudal Reinforcement Learning [1] and Federated Reinforcement Learning [2].

Feudal Reinforcement Learning employs a hierarchical structure wherein a *Manager-Agent* processes a global state to generate high-level goals. These goals are then decomposed into sub-goals assigned to *WorkerAgents* that operate in local environments. Such a modular approach not only simplifies problem-solving by breaking down complex tasks but also enhances scalability.

On the other hand, **Federated Reinforcement Learning** addresses the growing need for data privacy by allowing multiple client devices to update their models locally. These local models are periodically aggregated via federated averaging [2], ensuring that raw data remains on the device. The addition of FedProx regularization [3] minimizes discrepancies between local and global models, providing stability even in heterogeneous environments.

The integration of these paradigms is highly relevant in today's digital landscape, where the demand for scalable AI solutions and robust privacy-preserving methods is ever-increasing. This study is expected to contribute both theoretical insights and practical solutions, impacting fields such as healthcare, finance, and IoT.

1 Context and motivation

My motivation for undertaking this project stems from a longstanding interest in advanced machine learning techniques and their real-world applications. Being the study and applications for Multi-Agent Systems the primary reason to engage in this Master's degree. With previous experience in reinforcement learning and an enthusiasm for exploring novel paradigms, I am particularly drawn to the challenge of integrating hierarchical decision-making with federated

learning. The opportunity to develop a system that not only improves performance through task decomposition but also safeguards sensitive data is both intellectually stimulating and socially relevant. This project will allow me to deepen my expertise in AI while contributing to solutions that meet modern ethical and technical standards.

2 Goal

To evaluate the performance, scalability, and data privacy of Feudal Reinforcement Learning integrated to Federated Reinforcement Learning in multiagent environments in order to support the decompossition of complex tasks without compromissing privacy.

Secondary Goals

- Investigate the impact of hierarchical task decomposition on agent coordination and decision-making efficiency.
- Assess the effectiveness of federated averaging and FedProx regularization in maintaining consistency across distributed models.
- Evaluate the role of curriculum learning in gradually shifting agent reliance from true target data to manager-generated goals.
- Benchmark the proposed approaches using randomized simulation environments.
- Propose potential hybrid strategies that combine the strengths of both paradigms.

3 Sustainability, diversity, and ethical/social challenges

Sustainability The implementation of Federated Feudal Reinforcement Learning is primarily a software-driven project carried out in simulated environments, which results in a relatively modest direct ecological footprint. However, training deep learning models—particularly in reinforcement learning—can be computationally intensive, leading to high energy consumption and increased carbon emissions, especially when using large-scale distributed systems.

One of the key sustainability benefits of this project is its federated learning approach, which enables training across multiple local devices rather than relying on centralized cloud-based data centers. This distribution can reduce overall energy consumption, as it minimizes the need for massive data transfers and centralized GPU/TPU clusters,

which are known for their high power requirements. By keeping computation closer to the data source, the system aligns with the United Nations Sustainable Development Goals [4] —particularly SDG 9 (Industry, Innovation and Infrastructure) by promoting innovative, decentralized computing frameworks, and SDG 12 (Responsible Consumption and Production) by advocating for the efficient use of resources and the reduction of waste.

To ensure that the environmental impact of model training remains low, this project integrates CodeCarbon [5], a tool designed to measure, monitor, and optimize CO_2 emissions from machine learning experiments. CodeCarbon provides real-time feedback on the energy consumption of different computing hardware (CPUs, GPUs, TPUs) and estimates the carbon footprint of training sessions based on the geographical location of the data center or device.

Moving forward, sustainability assessments will remain integral to the project's optimization, ensuring that environmental impact is minimized while computational efficiency is maximized. The broader implications of such an approach include setting a precedent for energy-efficient AI research, demonstrating that advanced machine learning techniques can be both scalable and environmentally responsible.

Ethical behaviour and social responsibility A core feature of the project is its privacy-preserving approach, which keeps sensitive data on local devices rather than transmitting it to a central server. This strategy complies with current data protection laws and helps mitigate risks associated with data breaches. Although the research is technical, its outcomes have clear ethical benefits by fostering responsible data handling practices. In addition, this methodology may lead to the development of new industry standards in AI that emphasize social responsibility, ensuring that technological advances do not compromise personal privacy or security.

Diversity, gender and human rights While the project is predominantly technical and does not directly target issues of diversity or gender, its emphasis on secure, decentralized data processing contributes indirectly to the protection of human rights—specifically, the right to privacy. Unlike many machine learning applications that rely on centralized databases, this project does not use or maintain a database. As a result, it is not subject to data bias commonly found in datasets that may overrepresent or underrepresent specific demographic groups.

Since the learning process occurs in a decentralized manner, model training does not involve collecting, storing, or transferring sensitive user data. This approach minimizes risks related to biased data sampling, data leakage, and algorithmic discrimination. The

technology is designed to be neutral and universally applicable, with no inherent bias towards any gender or demographic group.

However, despite the lack of direct data collection, ensuring fairness remains crucial. Future deployments should incorporate fairness audits, model interpretability techniques, and robustness testing to confirm that the system operates equitably across diverse populations. Additionally, bias can still emerge through implicit biases in the problem formulation, reward functions, or decision-making processes within reinforcement learning systems. To address this, it is essential to adopt best practices in algorithmic fairness, such as regular testing with synthetic scenarios that account for diverse environmental conditions and agent behaviors.

By combining privacy-preserving methodologies with responsible AI practices, this project aims to support inclusive and accessible AI applications while respecting fundamental human rights, particularly the right to data privacy and protection against algorithmic discrimination.

4 Approach and methodology

The methodology for executing this project is centered on an agile, iterative development process that emphasizes continuous improvement and adaptive problem solving. Given the complexity of Reinforcement Learning implementations, and the need of extensive trial and error, our approach prioritizes the process of coding, testing and fine-tuning in an iterative manner. The key components of our methodology are outlined below:

Agile Development Process

- **Iterative Coding and Testing:** The project will follow an agile workflow [6] where code is written, tested, and refined in rapid, iterative cycles. Regular code reviews and testing (unit and integration) ensure that each iteration improves functionality and performance.

- **Rapid Prototyping and Adaptation:** New ideas and features will be quickly prototyped and evaluated [7]. This adaptive approach allows us to respond to challenges and make necessary adjustments early in the development process.

Development Environment and Tools

- **Python within the Anaconda Ecosystem:** [8] The project will be implemented in Python using the Anaconda distribution, ensuring a consistent and reproducible comput-

ing environment.

- **Integrated Development Tools:** We will use IDEs such as Spyder and Jupyter Notebook for efficient coding, debugging, and interactive experimentation.

- **Key Libraries and Frameworks:** The development leverages essential libraries including PyTorch [9] (for deep learning), Pygame [10] (for simulation rendering), Optuna [11] (for hyperparameter optimization), NumPy [12] (for numerical computation), PettingZoo [13] (for multi-agent environments), and CrazyRL [14] (as a base environment for reinforcement learning).

Documentation and Thesis Preparation

- **LaTeX for Thesis Writing:** The thesis document will be prepared using LaTeX [15], ensuring high-quality typesetting and adherence to academic standards. This approach facilitates clear presentation of the methodology, experiments, and outcomes.

- **Ongoing Documentation and Review:** Continuous documentation will be maintained throughout the project. Regular reviews will ensure that both the code and the accompanying thesis remain up-to-date, reflecting all iterations and insights gathered during the development process.

5 Schedule

The research plan (see Fig. 1.1) is organized into the following phases:

Phase 1: Literature Review and Theoretical Framework

- Review existing literature on Feudal Reinforcement Learning [1] and Federated Learning [2], as well as related works on FedProx [3] and other relevant sources.
- Define the theoretical basis for combining hierarchical and federated approaches.

Phase 2: System Design and Implementation

- Design and implement the neural network architectures for both ManagerAgent and WorkerAgents.
- Develop the federated averaging and FedProx [3] modules.
- Integrate the system into multiagent environments provided by the PettingZoo library [13].

Phase 3: Experimentation and Evaluation

- Run experiments with different architecutres and hyperparameters.
- Collect and analyze performance metrics including reward accumulation, convergence speed, and model consistency.
- Compare the performance of the individual paradigms and their hybrid implementations.

Phase 4: Documentation

- Document the process of the experimentaiton and evalutation.
- Translate results into insights.

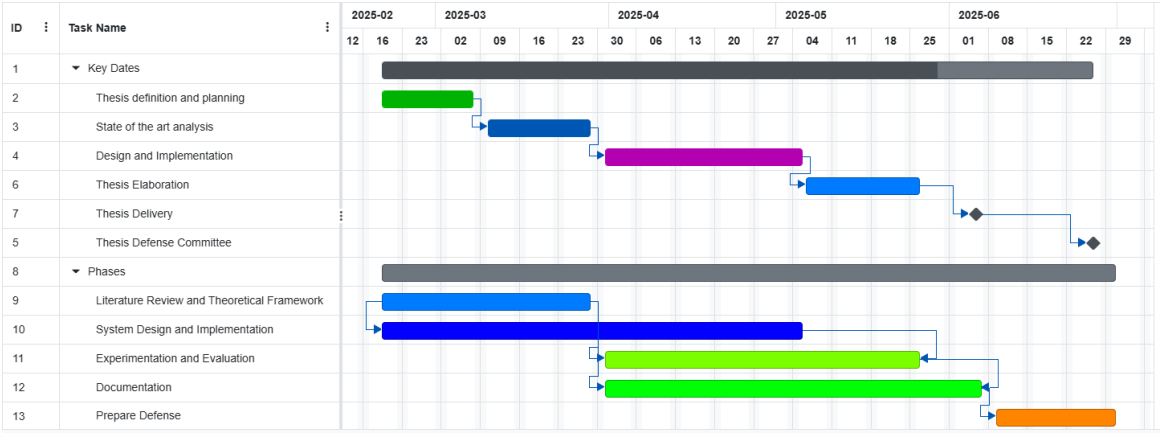


Figure 1.1: Gantt Project Plan.

6 Summary of the outputs of the project

- Synthesize experimental results and assess the impact of each approach.
- Identify potential improvements and future research directions.
- Document the methodology, experiments, and findings in a comprehensive final report.

7 Brief description of the remaining chapters of the report

This section outlines the structure of the remaining chapters, illustrating how each contributes to the overall objectives of the project.

- **Chapter 2 – State of the Art:** Reviews foundational concepts in federated learning, reinforcement learning, and multi-agent systems, along with recent advances in neural networks and privacy-preserving techniques.
- **Chapter 3 – Methods and Resources:** Describes the system design, including neural architectures, federated learning components, and simulation environments like PettingZoo.
- **Chapter 4 – Results:** Summarizes experimental results through performance metrics and visualizations, comparing configurations against project objectives.
- **Chapter 5 – Conclusions and Future Work:** Highlights key findings, limitations, and future directions, while reflecting on the project's broader ethical and societal implications.

Chapter 2

State of the Art

In this chapter, we provide a comprehensive overview of the theoretical foundations and state-of-the-art methodologies relevant to this study. The discussion is structured into the following main topics:

- Fundamentals of Reinforcement Learning (RL)
- Federated Reinforcement Learning (FRL)
- Hierarchical Learning Approaches
- Feudal Reinforcement Learning (Feudal RL)
- Historical Developments and Current Challenges
- Hyperparameter Tuning and Optimization in Reinforcement Learning

1 Reinforcement Learning

Reinforcement Learning (RL) is a framework where an **agent** learns to interact with an **environment** in order to maximize a cumulative reward. In this context:

- **Agent:** The decision-making entity that observes its surroundings (the state), selects actions based on a policy, and learns from the rewards or penalties received.
- **Environment:** The external system or world with which the agent interacts. The environment responds to the agent's actions by transitioning between states and providing feedback in the form of rewards.

This interaction is formalized as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R, γ):

- S: the set of states, representing all possible configurations of the environment.
- A: the set of actions, representing all possible decisions the agent can make.
- $P(s'|s,a)$: the state transition probability, describing the likelihood of transitioning to state s' when the agent takes action a in state s.
- $R(s,a)$: the reward function, which provides immediate feedback to the agent after taking an action in a given state.
- $\gamma \in [0,1)$: the discount factor, which weighs the importance of future rewards relative to immediate rewards.

At each time step, the process unfolds as follows:

1. **Observation:** The agent observes the current state $s \in S$ of the environment.
2. **Action:** Based on its policy π (as detailed in the next section), the agent selects an action $a \in A$.
3. **Transition and Feedback:** The environment transitions to a new state s' according to the probability $P(s'|s,a)$ and provides a reward $R(s,a)$.
4. **Learning:** The agent updates its policy based on the received feedback, with the aim of maximizing the cumulative reward over time.

This cycle of observation, action, and feedback is repeated continuously, enabling the agent to learn effective strategies even in complex and uncertain environments, as depicted in Fig. 2.1 .

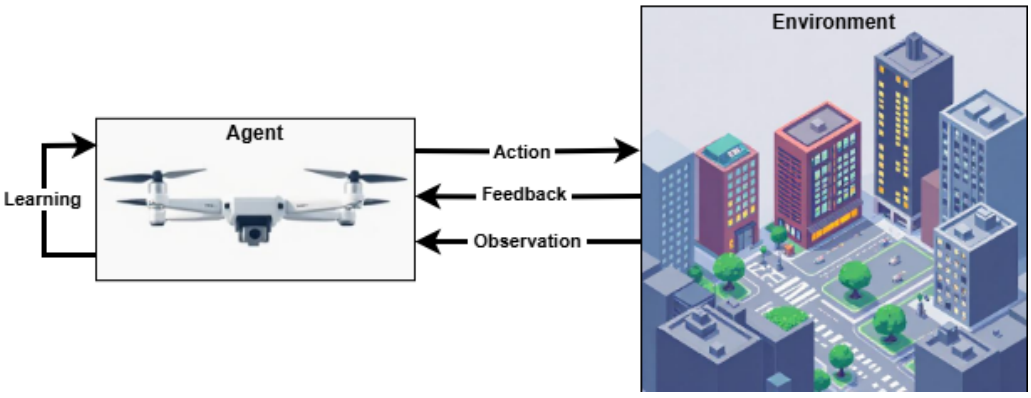


Figure 2.1: Environment - Actor interaction. Author's own work.

The main topics from RL that will be covered in this section:

1. The policy π.

2. The concept of value functions.

3. The Bellman equation and the optimal value function.

4. Q-learning as a foundational off-policy algorithm and its convergence properties.

5. Policy Gradient Methods that directly optimize the policy.

These topics not only form the theoretical backbone of RL but also provide the basis for many modern algorithms in both classical and deep reinforcement learning.

Policy

The policy, denoted by π, is the mechanism by which the agent selects actions based on the current state. The goal of most reinforcement learning algorithms is to find an optimal policy π^* that maximizes the expected cumulative reward over time. Whether the policy is directly parameterized (as in policy gradient methods) or derived indirectly from a value function (as in Q-learning), it defines the behavior of the agent in the environment.

Policies in reinforcement learning are typically categorized into two types:

- **Deterministic Policy:** A deterministic policy maps each state s to a specific action a with certainty. Formally, it is a function $\mu : S \to A$, where $\mu(s) = a$. This means that for any given state, the agent always takes the same action. Deterministic policies are common in algorithms such as Deep Deterministic Policy Gradient (DDPG), which operate effectively in continuous action spaces.

- **Stochastic Policy:** A stochastic policy assigns probabilities to actions given a state, represented as $\pi(a|s)$. This means the agent selects actions according to a probability distribution over the available actions in each state. Stochastic policies are particularly useful in environments where exploration is essential or when uncertainty in decision-making is beneficial. They are commonly used in classic policy gradient methods and entropy-regularized RL.

The choice between deterministic and stochastic policies depends on the problem context. Stochastic policies inherently incorporate exploration and are often favored in discrete or highly uncertain environments, while deterministic policies are typically more efficient in deterministic or continuous control settings.

Value Function

In reinforcement learning, the value function is a key concept that quantifies how desirable a state (or state-action pair) is, based on the expected cumulative rewards that an agent can obtain by interacting with the environment. It is defined as the *expected return* starting from a given state (or state-action pair) and following a particular policy thereafter. More formally, for an agent acting in an environment modeled as a Markov Decision Process (MDP), the state-value function is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s\right],$$

where:

- s is the current state,
- a_t are the actions taken by the agent,
- $R(s_t, a_t)$ is the reward received at time t,
- $\gamma \in [0, 1)$ is the discount factor that determines the present value of future rewards,
- $\mathbb{E}_{\pi}$ denotes the expectation over the trajectories generated by following the policy π.

Similarly, the action-value function $Q^{\pi}(s, a)$ extends the concept of the state-value function by quantifying not only the goodness of a state, but also the specific contribution of taking a particular action within that state. In essence, while the state-value function tells us the expected return (i.e., cumulative future reward) starting from state s and then following policy π, the action-value function

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s,\, a_0 = a\right]$$

provides a more fine-grained evaluation by determining the expected return when the agent starts in state s, takes action a immediately, and then continues to follow the policy π.

In practical terms:

- The state-value function $V^{\pi}(s)$ gives an overall measure of the "goodness" of being in a state s, aggregating the future rewards that can be accumulated from that state.
- The action-value function $Q^{\pi}(s, a)$ not only considers the state s but also captures the immediate effect of taking action a in that state. This is particularly useful because the agent often needs to compare the benefits of different actions available in the same state.

The relationship between these functions is evident in the way future rewards are considered. After taking action a in state s, the environment transitions to a new state s' according to the transition probability $P(s'|s,a)$, and the future return from s' is captured by $V^\pi(s')$. Thus, one can think of $Q^\pi(s,a)$ as the sum of the immediate reward and the discounted value of the next state.

With these concepts in mind, we now proceed to examine how the Bellman Equation formalizes the notion of value functions, setting the stage for the various algorithms used to optimize both the value function and the policy.

Bellman Equation

The Bellman Equation is a core concept in Reinforcement Learning which provides a **recursive** definition of the value of a state under a given policy. For a policy π, the state-value function is defined as:

$$V^\pi(s) = \mathbb{E}_\pi\Big[R(s,a) + \gamma V^\pi(s')\Big], \tag{2.1}$$

This captures the idea that the value of a state is the sum of the immediate reward and the discounted value of future states, averaged over actions and state transitions dictated by the policy.

Building on this, the **Optimal Value Function** defines the maximum achievable value from any state when following the best possible policy π^*:

The Optimal Value Function $V^{\pi^*}(s)$ takes this idea one step further. It represents the best possible value that can be obtained starting from state s, assuming that the agent follows the *optimal policy* π^*. The optimal value function is given by:

$$V^{\pi^*}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi^*}(s') \right\}. \tag{2.2}$$

This formulation seeks the action that maximizes the sum of immediate and expected future rewards, forming the theoretical basis for algorithms like value iteration and Q-learning.

Together, these equations express how optimal behavior can be achieved through recursive decision-making. A full explanation, including detailed diagrams and intuitive derivations, is provided in Annex 1.

Paradigms in Reinforcement Learning

Reinforcement Learning encompasses a variety of paradigms that differ in how agents learn from their environment, represent knowledge, and make decisions. These paradigms reflect high-level design choices that influence algorithm behavior, data efficiency, and policy optimization

strategies. Understanding these distinctions is essential for selecting or designing appropriate RL methods based on the characteristics of a specific problem or domain.

Value-based, Policy-based, and Actor-Critic Methods. Reinforcement learning methods are often categorized based on how the agent makes decisions and updates its behavior. In *value-based methods*, the learning process centers on estimating value functions—such as the state-value function $V(s)$ or the action-value function $Q(s, a)$. The policy is then derived implicitly by selecting the action that maximizes the estimated value. A canonical example of this approach is Q-learning, which we will discuss in this section.

In contrast, *policy-based methods* forgo value estimation and instead aim to directly learn a parameterized policy $\pi(a|s)$ that maximizes the expected return. These methods are particularly useful when dealing with high-dimensional or continuous action spaces.

Actor-Critic methods integrate the strengths of both approaches by maintaining two components: an actor, which updates the policy, and a critic, which evaluates the actions taken by estimating value functions. This hybrid strategy often leads to more stable and sample-efficient learning.

Model-free and Model-based Methods. Another key distinction lies in how the agent interacts with and learns about the environment. *Model-free methods* learn directly from observed experiences—state transitions and rewards—without constructing an explicit model of the environment's dynamics. These methods are generally simpler to implement and more robust to model inaccuracies, but they often require a larger number of interactions to learn effectively.

In contrast, *model-based methods* attempt to learn or are provided with a model of the environment, typically consisting of the transition probability function $P(s'|s, a)$ and the reward function $R(s, a)$. This model enables the agent to perform planning or simulate trajectories, thereby improving learning efficiency. While model-based approaches can be more sample-efficient, they may suffer from model bias if the learned model does not accurately reflect the true environment dynamics.

Exploration and Exploitation: A fundamental challenge in reinforcement learning is balancing *exploration*, the process of selecting less certain actions to acquire new knowledge about the environment, with *exploitation*, the act of leveraging current knowledge to maximize expected rewards. An agent must engage in exploration to avoid converging prematurely to suboptimal behaviors, while also exploiting known high-reward actions to ensure performance. Various reinforcement learning methods address this dilemma differently. For instance, value-based methods such as Q-learning often implement mechanisms like the ϵ-greedy strategy, where actions

are chosen randomly with some probability to promote exploration. Policy-based approaches, on the other hand, typically rely on inherently stochastic policies or incorporate entropy regularization to encourage behavioral diversity. Effectively managing this trade-off is critical for achieving sample-efficient and robust learning in uncertain or dynamic environments.

On-policy and Off-policy Learning. Reinforcement learning algorithms can also be differentiated by how the data used for learning relates to the policy being improved. *On-policy methods* evaluate and improve the same policy that is used to make decisions and interact with the environment. A representative example is the class of Policy Gradient Methods, where updates are based on the actions selected by the current policy, and learning is guided by the same policy being improved. We will discuss Policy Gradient Methods in this section as well.

In contrast, *off-policy methods* learn about a target policy using data generated from a different behavior policy. This decoupling allows for greater flexibility in data collection and reuse. Q-learning exemplifies this approach, as it learns the value of the *optimal policy*—the policy that selects the best possible action in each state—regardless of the policy that is actually used to generate the data.

Q-learning

Q-learning is an off-policy, value-based, model-free reinforcement learning algorithm that learns the action-value function $Q(s, a)$ using the update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big),$$

where α is the learning rate.

The equation updates our estimate of the value of taking action a in state s by moving it slightly towards the observed return. The term $R(s,a) + \gamma \max_{a'} Q(s',a')$ represents the *target* value, which is the sum of the immediate reward $R(s,a)$ received after taking action a and the best possible (i.e., highest) discounted value that can be obtained from the next state s'. The difference between this target and the current estimate $Q(s,a)$ is called the *temporal difference error*. Multiplying this error by the learning rate α controls how aggressively the estimate is updated. Under certain conditions, Watkins and Dayan [21] proved that this update rule converges to the optimal action-value function $Q^{\pi^*}(s,a)$, meaning that as learning progresses, the estimates become closer to the true values needed for optimal decision making.

Policy Gradient Methods

Policy Gradient Methods directly optimize the policy by maximizing the expected return, making them an on-policy and model-free class of reinforcement learning. Being *model-free*, they

do not require an explicit model of the environment's dynamics, learning instead from sampled interactions. As *on-policy* methods, they improve the very policy that is used to generate behavior, updating it based on the returns observed while following it. Often implemented with stochastic policies that naturally encourage exploration through randomization in action selection. The policy gradient theorem states:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a|s)\, Q^{\pi_\theta}(s,a)\Big]. \tag{2.3}$$

This theorem can be understood as follows: Instead of trying to learn a value function and then indirectly deriving the policy from it, policy gradient methods directly adjust the policy parameters θ in the direction that increases the expected return $J(\theta)$. The gradient $\nabla_\theta J(\theta)$ tells us how to change θ so that actions that lead to higher returns become more likely. In this equation, $\nabla_\theta \log \pi_\theta(a|s)$ represents the sensitivity of the policy's probability of taking action a in state s with respect to its parameters. Multiplying this by the action-value $Q^{\pi_\theta}(s,a)$ (which estimates the quality of taking action a in state s) gives a measure of how beneficial that action was. Taking the expectation over the distribution of actions and states under the current policy π_θ aggregates these effects, providing a direction for updating θ. This formulation forms the basis of actor-critic algorithms where an *actor* (the policy) is updated using feedback from a *critic* (the value function) [20].

Literature Review

Early works in RL, including the foundational texts by Bellman [19] and the influential book by Sutton and Barto [20], laid the groundwork for modern approaches. The advent of deep learning gave rise to Deep Q-Networks (DQN) [22] which marked a significant milestone by successfully applying RL to high-dimensional sensory inputs. However, challenges such as instability and divergence in training spurred the development of more robust methods like the actor-critic framework and Proximal Policy Optimization (PPO) [23]. Current challenges remain in dealing with sparse rewards and sample inefficiency, motivating continued research into more stable and data-efficient methods.

2 Federated Reinforcement Learning

Federated Reinforcement Learning (FRL) extends RL to a distributed setting where multiple agents learn from decentralized data without sharing raw information. In this paradigm, each client trains a local RL agent, and periodically the models are aggregated using methods like

federated averaging:

$$\theta^{(t+1)} = \frac{1}{K} \sum_{k=1}^{K} \theta_k^{(t+1)},$$

where:

- $\theta_k^{(t+1)}$ represents the updated local model parameters from client k after local training during round $t+1$,
- K is the total number of participating clients,
- $\theta^{(t+1)}$ is the resulting global model after averaging.

This aggregation rule effectively computes the element-wise mean of all client updates, enabling collaborative learning without exposing private data. It assumes that all clients contribute equally and that their data distributions are not overly divergent [2].

However, in practice, data on different clients is often non-identically distributed (*non-iid*), which can lead to inconsistent local updates and poor global convergence. To address this challenge, the *FedProx* method introduces a regularization term into the local objective:

$$L_k(\theta) = f_k(\theta) + \frac{\mu}{2} \|\theta - \theta^{(t)}\|^2,$$

where:

- $f_k(\theta)$ is the empirical loss function on the local data of client k,
- θ is the current set of parameters being optimized locally,
- $\theta^{(t)}$ is the global model from the previous communication round t,
- μ is a non-negative hyperparameter controlling the strength of the proximal term.

The proximal term $\frac{\mu}{2} \|\theta - \theta^{(t)}\|^2$ penalizes local models that deviate significantly from the previous global model, thus promoting stability and reducing the effect of heterogeneity across clients. By constraining how far each local model can drift, FedProx improves convergence and model consistency across the federation [3].

Figure 2.2 presents the architecture of Federated Reinforcement Learning. Each client (or agent) interacts with its local environment and trains a local model. Periodically, these local models are sent to a central aggregator, which performs federated averaging to update the global model without sharing raw data.

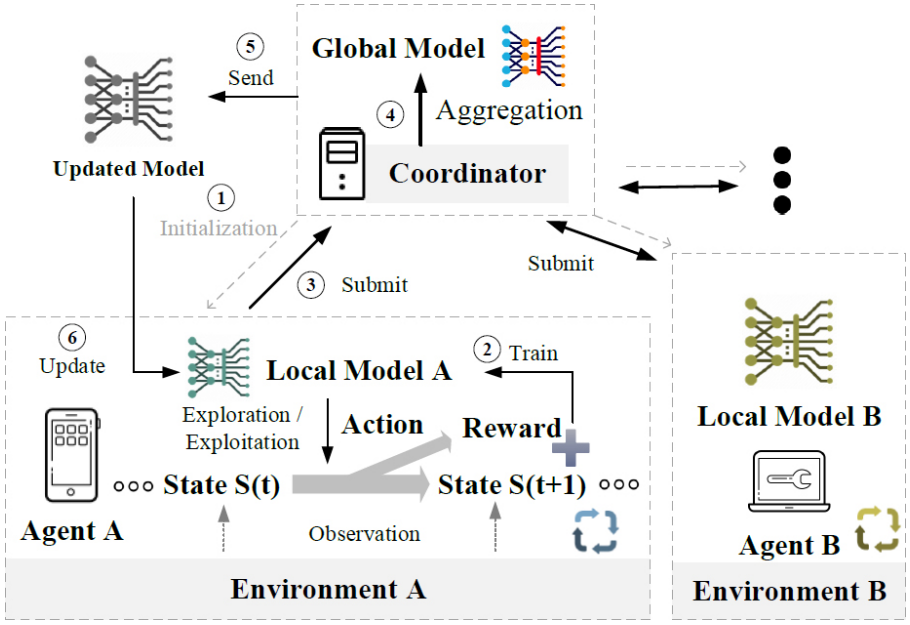


Figure 2.2: Federated RL with client-side learning and central model aggregation. Adapted from [2].

Literature Review

Federated learning was popularized by McMahan et al. [2] in the context of mobile devices, emphasizing privacy and communication efficiency. Subsequent research has focused on addressing statistical heterogeneity and communication challenges [3]. In reinforcement learning, extending these ideas introduces additional complexities due to the exploration–exploitation trade-off. The combination of decentralized data and sequential decision making has been explored in recent works [24] that adapt actor-critic and policy gradient methods to federated settings. Current research is actively addressing issues such as model drift and non-iid data distributions, making FRL a vibrant and evolving area.

3 Population Based Training

Population Based Training (PBT) offers an alternative to federated averaging by introducing a population-level evolutionary strategy that selects and mutates high-performing agents over communication rounds. Instead of averaging all local models into a single global policy, PBT maintains a diverse population of agents across clients, evaluates their performance periodically, and promotes the best-performing ones through replication and mutation.

In the PBT paradigm, each client trains its agent independently with its own policy. After a training round, agents are ranked based on performance (e.g., accumulated reward). The worst-performing agents replace their policies with those of better-performing peers, optionally

applying small perturbations to encourage exploration:

$$\theta_k^{(t+1)} = \theta_{k'}^{(t)} + \Delta, \quad \text{where } k' = \arg\max_j R_j^{(t)} \text{ and } \Delta \sim \mathcal{N}(0, \sigma^2),$$

where:

- $\theta_k^{(t+1)}$ is the updated policy of client k,
- $R_j^{(t)}$ is the performance of agent j at round t,
- Δ is a small stochastic perturbation applied to encourage diversity,
- σ is the perturbation scale.

This evolutionary selection process helps guide the population toward stronger policies without assuming model similarity or balanced data distributions. Unlike federated averaging, which blurs agent-specific behaviors, PBT encourages specialization and leverages client heterogeneity by allowing different learning dynamics and parameter schedules.

Figure 2.3 illustrates this process. Clients maintain independent policies, share performance metrics, and occasionally sync to better-performing peers while exploring new configurations.

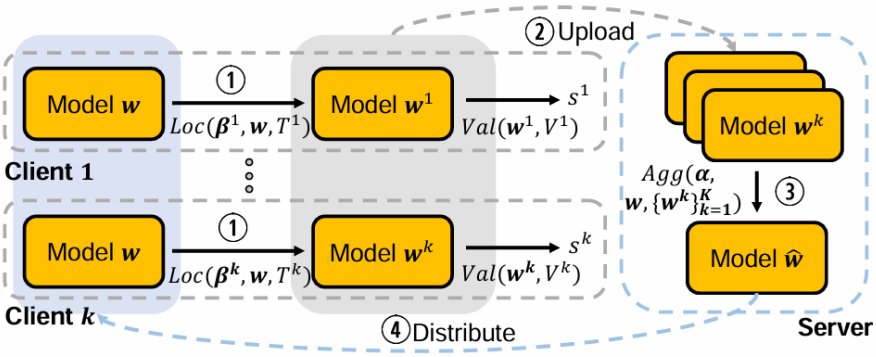


Figure 2.3: Population Based Training round update in a federated setting. Adapted from [26]

Literature Review

Population Based Training was introduced by Jaderberg et al. [25] to automate hyperparameter tuning and exploit/perturb strategies in deep reinforcement learning. It has since been extended to distributed settings and applied to heterogeneous learning scenarios. In federated contexts, population-based training methods have demonstrated improved robustness to variations in client data distributions—where each client may observe different environments or reward structures—and offer an effective alternative to naive parameter averaging, which can lead to degraded performance in such heterogeneous settings [26].

4 Hierarchical Learning Approaches

Hierarchical Learning decomposes complex decision-making tasks into multiple levels of abstraction. The idea is to have:

- A high-level controller (or manager) that selects abstract goals or sub-tasks.
- Low-level controllers (or workers) that execute primitive actions to achieve these sub-goals.

As depicted in Figure 2.4, hierarchical reinforcement learning introduces two levels of abstraction. A high-level manager selects sub-goals or tasks, while a lower-level controller executes primitive actions to fulfill these goals. This decomposition enables structured decision making over long time horizons.

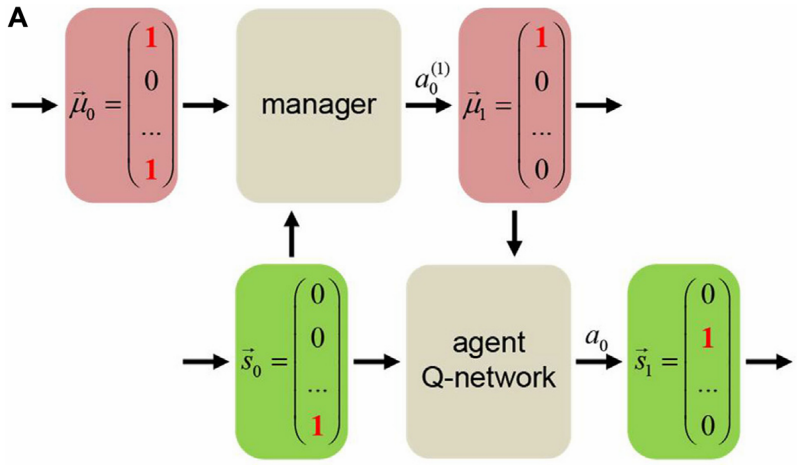


Figure 2.4: Two-level abstraction in hierarchical reinforcement learning. Adapted from [27].

A mathematical formulation of a hierarchical policy can be written as:

$$\pi(a|s) = \sum_{g \in \mathcal{G}} \pi^M(g|s)\pi^W(a|s,g), \tag{2.4}$$

where $\pi^M(g|s)$ is the manager's policy over goals g and $\pi^W(a|s,g)$ is the worker's policy conditioned on the goal. This formulation, which integrates the decision-making processes at multiple levels of abstraction, is discussed in detail in the options framework [28] and has been further refined in hierarchical deep reinforcement learning approaches [29].

Proof Sketch: Hierarchical Bellman Equation

Consider the state-value function under a hierarchical policy:

$$V(s) = \max_{g \in \mathcal{G}} \{Q(s, g)\},$$

with the option-value function defined as:

$$Q(s, g) = \mathbb{E}\left[\sum_{t=0}^{\tau-1} \gamma^t r(s_t, a_t) + \gamma^\tau V(s_\tau) \,\middle|\, s, g\right],$$

where τ is the duration until the goal g terminates. This formulation generalizes the standard Bellman equation to a hierarchical setting (see [28] for a detailed derivation).

Literature Review

The hierarchical framework dates back to early work on the options framework [28] and has been further refined by approaches such as hierarchical deep RL. In particular, Kulkarni et al.[29] advanced the field by introducing a two-level architecture that decouples high-level decision making from low-level control through the automatic generation of sub-goals, which effectively guide behavior over long horizons. Their integration of intrinsic rewards to foster exploration helped address the limitations posed by sparse extrinsic feedback, thereby enhancing sample efficiency and enabling more structured exploration. Despite these significant contributions, challenges still remain in reliably discovering effective hierarchies and ensuring efficient credit assignment across levels. Recent work continues to focus on refining sub-goal discovery and balancing intrinsic with extrinsic rewards to drive robust learning.

5 Feudal Reinforcement Learning

Feudal Reinforcement Learning (Feudal RL) is a specialized hierarchical approach proposed by Vezhnevets et al. [1], designed to address complex tasks that benefit from temporal abstraction and modular policy structure. Inspired by the concept of feudal organization, the framework introduces a hierarchy of control between two types of agents:

- The **ManagerAgent** operates at a higher level of abstraction. It observes a global or abstracted representation of the environment state s, and emits sub-goals $g = \pi^M(s)$ that describe desirable directions or objectives for lower-level agents.

- The **WorkerAgents** operate at a finer level of granularity. Each worker receives a goal g from the manager and its local state (or full state, depending on the implementation), and selects low-level actions $a = \pi^W(s, g)$ intended to fulfill the sub-goal.

Figure 2.5 shows the Feudal Reinforcement Learning framework. A ManagerAgent produces high-level directional goals based on the global state. These are passed to WorkerAgents that act in their local environments to achieve the sub-goals. This hierarchy enables scalable and modular learning in complex tasks.

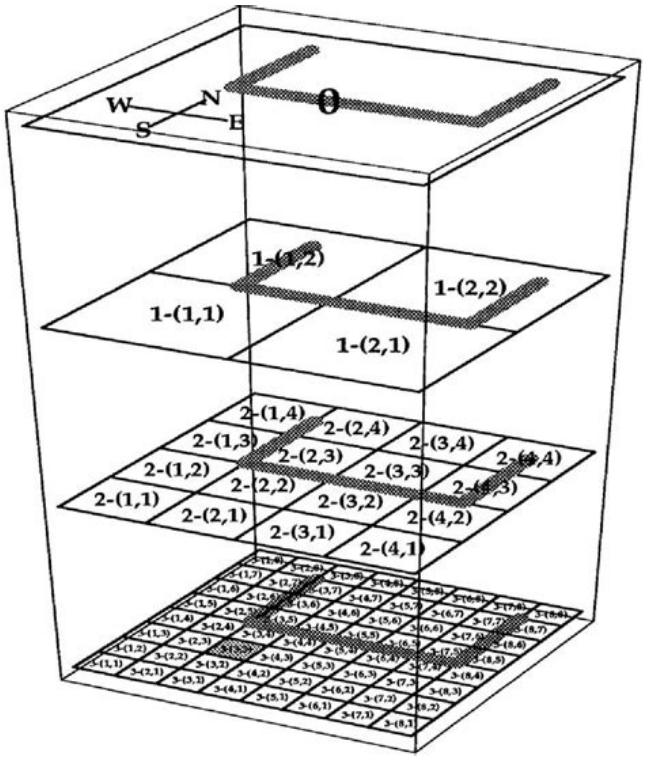


Figure 2.5: Feudal RL structure with manager and worker agents. Adapted from [1].

Training such hierarchical agents presents additional challenges compared to flat architectures. The most prominent issue is **credit assignment across layers**: if the overall task succeeds or fails, how much of the outcome should be attributed to the manager's goal selection versus the workers' execution? To mitigate this, Vezhnevets et al.[1] introduced a specialized loss function known as the **consistency loss**, which encourages coherence between the manager's generated goals and the actual progress made by the workers.

$$L_{\text{consistency}} = \frac{1}{N} \sum_{i=1}^{N} \|g_i - (t_i - s_i)\|^2, \tag{2.5}$$

where t_i is the target state and s_i is the current state for agent i.

This mechanism also facilitates goal generalization, as the manager learns to encode navigational intents or task-relevant transitions without requiring direct access to reward signals. Workers, in turn, are trained to interpret and fulfill these goals via their own action-value functions or policy networks, depending on the implementation.

Feudal RL has been successfully applied in high-dimensional environments such as navigation and manipulation tasks, where decomposing behavior into abstract planning and fine-grained control yields better generalization and learning stability. Despite its advantages, further research is needed to refine sub-goal discovery, improve inter-agent coordination, and reduce training overhead.

Mathematical Derivation and Analysis

Consider a multi-agent environment where each worker's state s_i and the target t_i are known. The manager produces a goal g_i intended to guide the worker toward the target. The derivation of the consistency loss ensures that the goal approximates the vector difference:

$$g_i \approx t_i - s_i.$$

Minimizing $L_{\text{consistency}}$ via gradient descent helps align the manager's predictions with the desired sub-goals, thereby improving coordination between manager and workers. This mechanism is central to stabilizing learning in complex environments with multiple agents.

Literature Review

Feudal RL builds on the broader principles of hierarchical RL but introduces a modular structure that is particularly suited for environments where tasks can be decomposed naturally into sub-tasks. Vezhnevets et al. [1] demonstrated that this architecture can lead to improved performance in high-dimensional control tasks. However, feudal reinforcement learning faces several challenges that remain active research topics. One major issue is the efficient credit assignment between the high-level manager and low-level workers: it can be difficult to determine how much of the eventual reward should be attributed to the manager's abstract sub-goals versus the worker's specific actions. Additionally, ensuring that the manager's sub-goals are both meaningful and achievable by the workers is non-trivial, particularly in environments with sparse extrinsic rewards. While the seminal work by Vezhnevets et al. laid the foundation nearly a decade ago, subsequent research in hierarchical RL—often within the broader context of temporal abstraction and sub-goal discovery, as seen in works like [28] and [29]—has sought to address these issues. Although there have not been many new papers that build directly on

the original feudal RL framework, the challenges it exposes continue to inspire further investigation into better mechanisms for goal generation, intrinsic motivation, and the integration of hierarchical structures within federated and multi-agent settings.

6 Historical Developments and Current Challenges

The evolution of RL can be traced from early dynamic programming methods introduced by Bellman [19] to the modern deep RL algorithms that combine function approximation with massive computational resources. Early breakthroughs in RL laid the foundation for later methods such as DQN [22] and actor-critic approaches [20].

In the realm of federated learning, the pioneering work by McMahan et al. [2] has spurred research into privacy-preserving algorithms that address communication efficiency and statistical heterogeneity. FedProx [3] further refines this by incorporating a proximal term to stabilize local updates.

Current Challenges include:

- **Scalability:** Both in terms of learning complex policies and handling large-scale federated systems. Federated reinforcement learning must scale to numerous clients with diverse data sources, demanding efficient communication protocols [2], while hierarchical methods require coordination among multiple levels [1].

- **Data Heterogeneity:** Ensuring robust performance when client data distributions differ significantly. Non-IID data in federated settings may cause instability, and aligning subgoals in a hierarchical model across varying environments adds further complexity [3].

- **Credit Assignment:** In hierarchical settings, properly attributing rewards across multiple layers remains a difficult problem. This is compounded in federated settings where delayed and aggregated feedback makes it challenging to discern the contribution of local updates versus global improvements.

- **Stability and Convergence:** Deep RL models can experience divergence if not carefully managed. In federated scenarios, asynchronous updates and model divergence across clients further complicate the convergence process.

Ongoing research addresses these challenges through novel architectures, improved optimization techniques, and enhanced theoretical guarantees. The convergence of hierarchical, federated, and deep reinforcement learning represents a promising yet challenging frontier that continues to evolve rapidly.

Potential Integration: Federated Feudal Reinforcement Learning

A potential approach to merge federated reinforcement learning with feudal reinforcement learning is to develop a *Federated Feudal Reinforcement Learning* (FFRL) framework. In such a system, each local device would run a hierarchical model comprising:

- A **ManagerAgent** that generates high-level sub-goals based on the global state.
- **WorkerAgents** that execute low-level actions conditioned on these sub-goals.

Local models could be periodically aggregated using techniques like federated averaging [2] and proximal regularization [3] to maintain consistency across clients.

This integration offers several advantages:

- **Data Privacy:** Sensitive data remains on local devices.
- **Task Decomposition:** Hierarchical decomposition facilitates efficient learning on complex tasks.
- **Modularity:** The separation of high-level and low-level decision making allows flexible adaptation in heterogeneous environments.

However, significant challenges remain:

- **Synchronization:** Aligning high-level strategies (sub-goals) across clients with different local environments.
- **Communication Overhead:** The added layer of hierarchical coordination may increase communication requirements.
- **Complex Credit Assignment:** Developing methods to correctly attribute rewards both within each hierarchical model and across federated updates.

Continued empirical and theoretical research is essential to ensure that the benefits of combining these paradigms outweigh the additional complexities introduced.

Literature Review

The foundation of modern reinforcement learning (RL) is built on several key contributions from the literature. Bellman's pioneering work on dynamic programming [19] laid the theoretical groundwork, while Sutton and Barto's comprehensive treatment of RL [20] established many of the core principles, including value functions and policy gradients. Watkins and Dayan's introduction of Q-learning [21] further advanced the field by providing an off-policy algorithm with provable convergence properties. The success of deep RL is largely attributed to the

work by Mnih et al. [22], who demonstrated the application of deep Q-networks (DQN) in high-dimensional environments.

In parallel, the evolution of federated learning has addressed challenges such as data heterogeneity and communication efficiency. The work by McMahan et al. [2] introduced federated averaging (FedAvg), a method that enables decentralized training, while Li et al. [3] proposed FedProx to stabilize updates in the presence of non-IID data. These contributions underscore the complexity of scaling RL to decentralized settings and highlight ongoing challenges in credit assignment, stability, and convergence.

7 Hyperparameter Tuning and Optimization in Reinforcement Learning

The discussion in previous sections has outlined the significant challenges faced by modern reinforcement learning (RL) approaches, including scalability, data heterogeneity, credit assignment, and stability. These challenges not only complicate the design of federated and hierarchical RL systems but also greatly influence the choice of hyperparameters. In practice, deep RL algorithms are extremely sensitive to settings such as learning rate, discount factor, and entropy coefficients. As a result, the process of hyperparameter tuning becomes critical for achieving convergence and robust performance.

Recent advances in automated optimization frameworks have brought tools like **Optuna** [11] to the forefront of RL research. By leveraging techniques such as Bayesian optimization and adaptive sampling, Optuna and similar tools enable efficient exploration of vast hyperparameter spaces that would be infeasible to search manually. These frameworks not only reduce the manual trial-and-error process but also help uncover non-intuitive yet highly effective configurations.

Figure 2.6 shows a parallel coordinates plot from Optuna, visualizing hyperparameter search. Each line represents a trial, and its color reflects the corresponding objective value. This visualization helps identify parameter interactions and trends that lead to better performance.

This focus on hyperparameter optimization is directly derived from the challenges discussed earlier. For instance, the stability issues in federated settings and the difficulties in credit assignment within hierarchical models underscore the need for precise tuning.

Hyperparameter tuning is a key component in bridging theoretical advancements with practical implementations in RL. Tools like Optuna have emerged as state-of-the-art solutions, ensuring that complex models are not only theoretically sound but also optimized for real-world performance.

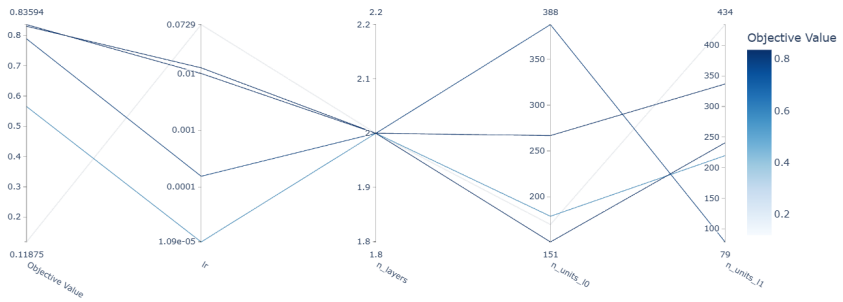


Figure 2.6: Optuna visualization of hyperparameter tuning trials. Adapted from [11].

Literature Review

Hyperparameter tuning is a critical aspect of training modern RL algorithms, which are notably sensitive to settings such as learning rate, discount factor, and entropy coefficients. The vast hyperparameter space and the non-stationarity of RL environments make manual tuning impractical. Early work by Bergstra and Bengio [30] demonstrated that random search can outperform grid search in many scenarios, setting the stage for more advanced optimization techniques.

Subsequent studies, such as those by Snoek et al. [31], explored Bayesian optimization methods to automate hyperparameter tuning. More recently, frameworks like Optuna [11] have emerged as state-of-the-art tools, leveraging adaptive sampling and Bayesian optimization to efficiently explore hyperparameter configurations. These approaches not only reduce manual effort but also uncover non-intuitive parameter settings that can significantly enhance RL performance.

8 Summary

The methodologies reviewed in this chapter—from the basic MDP formulation and Bellman equations to advanced topics like federated and feudal reinforcement learning—demonstrate the breadth and depth of current research in RL. Detailed mathematical derivations, such as those of the Bellman equation (5.1) and the policy gradient theorem (2.3), provide the theoretical underpinnings for these methods. Historical developments have paved the way for modern approaches that address real-world challenges in scalability, data privacy, and complex decision-making.

By integrating these state-of-the-art techniques, our proposed framework seeks to leverage the strengths of each approach to tackle contemporary challenges in multi-agent environments while maintaining robustness and privacy.

Chapter 3

Methods and Resources

In this chapter we describe the design and implementation choices underlying our Federated Feudal RL prototype, the methodology followed during development, and the software artifacts produced.

1 Problem Statement and Simulation Environment

1.1 Implementation Environment

- **Language & Libraries:** Implemented in Python 3.11 using PyTorch [9] for neural network modeling and optimization, Optuna [11] for hyperparameter tuning, and NumPy [12] for numerical data handling.

- **Simulation Environment:** Experiments were conducted using the `CrazyRL` [14] framework, specifically the `Catch` environment, which provides a simple multi-agent setup suitable for evaluating hierarchical and federated learning strategies. The environment leverages Pygame [10] to visualize agent behavior and interactions during simulation runs.

1.2 Problem Statement.

This work addresses the challenge of performing privacy-preserving multi-agent reinforcement learning using a Federated Feudal Reinforcement Learning (FedFeudRL) approach. Our objective is to train decentralized drone agents in a shared task, where each client operates in its own private simulation environment without directly exchanging local experiences or gradients. The agents operate in a multi-agent setting using the `Catch` environment from the CrazyRL library [14], integrated with the PettingZoo API [13]—a standardized multi-agent extension.

The agents are organized into clients, each running local feudal actor-critic policies. A central manager computes long-term spatial goals, while local worker agents derive short-term actions from these goals. The framework applies federated averaging (FedAvg) for global synchronization

1.3 Catch Environment.

`Catch` simulates a team of drone agents collaboratively trying to catch a dynamically escaping target in 3D space. The environment ensures realistic drone behavior and includes basic collision physics. The target avoids agents by moving in the opposite direction of the drones' centroid, with random perturbations and boundary repulsion near edges. See Fig. 3.1 for a visual representation.

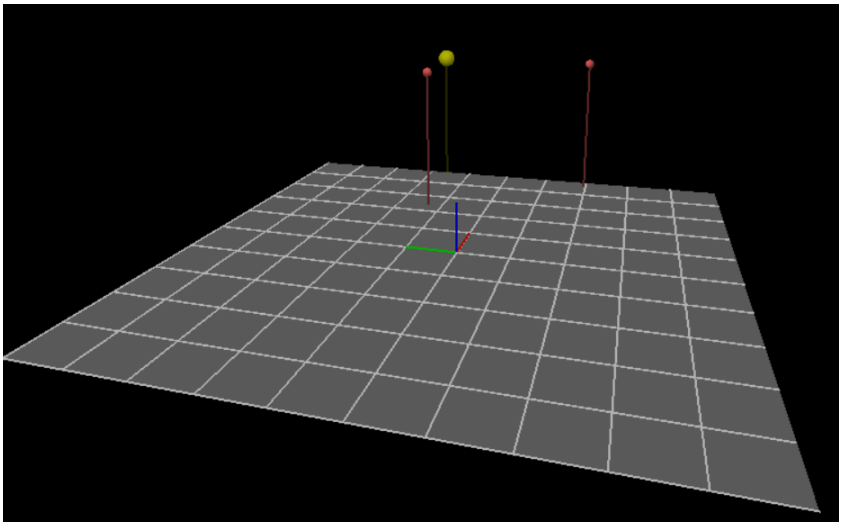


Figure 3.1: Example of human render mode in the Catch environment using CrazyRL's Pygame. Red agents attempt to capture the yellow target. CrazyRL Simulation [14].

Each interaction within the environment is organized into an *episode*, which represents a complete trajectory of the agents and target over a fixed number of discrete *steps*. At each step, all agents simultaneously select continuous 3D movement vectors (actions), scaled to a maximum movement of 20 cm per axis. The target also updates its position at each step, moving at a speed of 10 cm per axis.

The target's motion is governed by predefined escape dynamics: it moves away from the mean position (centroid) of all agents. If the agents are sufficiently far, the target moves directly in the opposite direction of their centroid. However, when agents get too close (within a set threshold), the target begins to jitter with small random movements, simulating hesitation or panic. Additionally, a soft repulsion is applied when the target nears the environment's

boundaries, nudging it back toward the center and preventing it from getting trapped at the edges.

An episode terminates under one of three conditions:

- **Capture:** Any agent enters a small proximity threshold around the target, successfully "catching" it.
- **Crash:** An agent collides with another agent or the ground (i.e., its altitude falls below a safe threshold).
- **Step limit:** The episode reaches a predefined maximum number of steps (e.g., 50), after which it is truncated regardless of success or failure.

Observation Space. The observation space defines what each agent perceives at every timestep and serves as the input to both local policy networks and the global manager. It encodes spatial awareness necessary for agents to navigate and coordinate in 3D space while tracking a moving target.

Each agent observes a flat vector consisting of:

- Its own 3D position (x, y, z).
- The 3D position of the target.
- The 3D positions of all other agents.

The shape of the observation vector is $3\times(N+1)$, where N is the number of agents. All positions are bounded within $[-\text{size}, \text{size}]$ in x-y and $[0, 3]$ in z as per the simulation environment limits.

Manager Observation. Unlike the workers, the manager does not directly interact with the environment. Instead, it constructs a global observation from worker-provided data. This vector includes:

- The 3D positions of all N worker agents.
- The 3D position of the target (shared across agents).
- The estimated 3D velocity of the target, computed as the difference between its current and previous position.

The resulting manager state vector has dimension $3N + 3 + 3 = 3(N + 2)$, where the last three entries correspond to the target's velocity. This allows the manager to output temporally extended spatial goals for each agent, considering both current configuration and target dynamics.

Action Space. The action space defines the set of possible outputs an agent can produce at each timestep to interact with the environment. In our hierarchical architecture, both worker and manager agents produce actions, but their semantics differ.

Worker Agent Actions. Each worker agent outputs a continuous 3D action vector representing the desired movement direction along the x, y, and z axes. The action space for each worker is defined as:

$$\mathcal{A}_i = [-1, 1]^3 \quad \text{for each agent } i.$$

These raw actions are scaled to represent a maximum of 20 cm movement per timestep and clipped to ensure the agent remains within the simulation boundaries. The worker uses both its local position and the goal assigned by the manager to generate these actions.

Manager Agent Output. In contrast, the manager agent does not directly act in the environment. Instead, it produces temporally extended spatial goals for each worker. These goals also lie in 3D space and guide the behavior of workers over multiple steps. Formally, for N agents, the manager outputs N separate 3D goal vectors:

$$\mathcal{G} = \{g_1, g_2, \ldots, g_N\}, \quad g_i \in \mathbb{R}^3.$$

Each goal g_i serves as a high-level directional signal for worker i, and is sampled from a distribution predicted by the manager's policy network. The x and y components are bounded by the environment size (via a scaled tanh), while the z component is scaled to lie within a valid altitude range (e.g., $[0.2, 3.0]$).

This division of action roles enables hierarchical control: the manager plans long-term coordination strategies while workers execute fine-grained control to achieve those goals.

Given the extensive empirical tests conducted on the reward function design, its details are not included under the environment specification but are instead discussed in the system architecture section (see Subsection 2.2).

2 Design and Development

All source code and implementation details are available in the project repository [32].

2.1 System Architecture

- **Global Manager–Local Worker Split.** We adopt a Feudal Reinforcement Learning paradigm in which a centralized `ManagerAgent` periodically generates optimized high-level goals for multiple decentralized `WorkerAgents`, each operating in its own local environ-

ment (See Fig. 3.2). This architectural split enables temporal abstraction and modular policy learning.

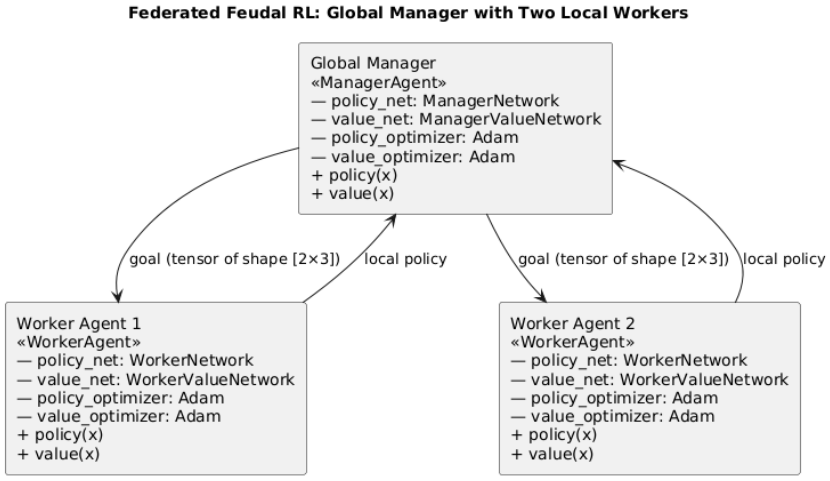


Figure 3.2: Manager-Worker Interaction. Author's own work.

The `ManagerAgent` observes an abstract global state—comprising the spatial positions of all worker agents and the target—and generates a goal every three environment steps. These goals are sampled from a learned multivariate Gaussian policy and represent optimized directional objectives for each worker. This temporal decoupling allows the manager to focus on high-level coordination, such as goal-setting and trajectory alignment, rather than fine-grained control.

Each `WorkerAgent` receives its individual goal vector from the manager (updated once every three steps) and combines it with its own local observation (its 3D position). The worker's objective is to execute low-level actions that move it toward the target provided by the manager, effectively interpreting the goal as a moving objective to be captured. Workers are trained via actor-critic methods using only local observations and intrinsic reward signals based on proximity to the actual target.

The communication between manager and workers is minimal:

- From manager to worker: a 3D goal vector representing the optimized sub-target location.
- From worker to manager: no direct feedback, only model updates during training.

While the global `ManagerAgent` constructs a centralized abstract state composed of agent and target positions for high-level decision-making, the federated training protocol itself maintains strict privacy boundaries. Clients never transmit raw trajectories, sensor data, or environment logs across the federation. Only model parameters are exchanged, and

all environment interactions remain local to each client. This preserves privacy among clients and aligns with the principles of decentralized learning, even though centralized coordination is retained at the planning level.

- **Learning Algorithm – Advantage Actor-Critic (A2C).** Both the `ManagerAgent` and each `WorkerAgent` are trained using the Advantage Actor-Critic (A2C) algorithm, a synchronous policy gradient method that combines value estimation with direct policy optimization. In this framework, each agent maintains two networks: a policy (actor) that outputs either movement actions (for workers) or spatial goals (for the manager), and a value function (critic) that estimates expected returns from the current state.

  As mentioned previously, for the `WorkerAgent`, the input to these networks is a concatenation of its local 3D position and the goal vector received from the manager. For the `ManagerAgent`, the input is a centralized global observation composed of all agent positions, the target position, and the estimated target velocity.

  The A2C framework provides several benefits:

  - **Sample efficiency:** By using advantage estimates (via Generalized Advantage Estimation, GAE), the variance of policy gradients is reduced, stabilizing training.
  - **Modular learning:** Separating policy and value networks enables flexible exploration strategies while retaining stable value estimation.
  - **On-policy compatibility:** A2C is well-suited to on-policy training, which matches the episodic and decentralized nature of our federated setup.

  However, A2C also has limitations in distributed settings:

  - **On-policy constraint:** It cannot reuse old experiences, potentially increasing sample requirements per episode.
  - **Sensitivity to noise:** Because updates rely heavily on immediate observations, high stochasticity in environments (e.g., randomized initial positions) may affect convergence speed.

  Despite these trade-offs, A2C strikes an effective balance between complexity and performance, making it suitable for multi-agent and hierarchical architectures under constrained privacy assumptions.

- **Federated Setting.** The system operates under a federated learning paradigm with four decentralized clients. Each client simulates a separate environment instance and hosts two `WorkerAgent`s, both sharing a single neural-network model for efficiency and weight

coherence. During training, each client independently runs multiple feudal episodes, during which the shared worker model is updated locally using actor-critic gradients and FedProx regularization (See Fig. 3.3).

At the end of each communication round—comprising a fixed number of local episodes per client—the worker models are aggregated using *Federated Averaging* (FedAvg). Specifically, the parameters of the local worker networks (both policy and value) are averaged element-wise across all clients to produce a new global model. This model is then redistributed to all clients to initialize the next training round, ensuring that all worker agents begin each round from a shared, synchronized set of parameters.

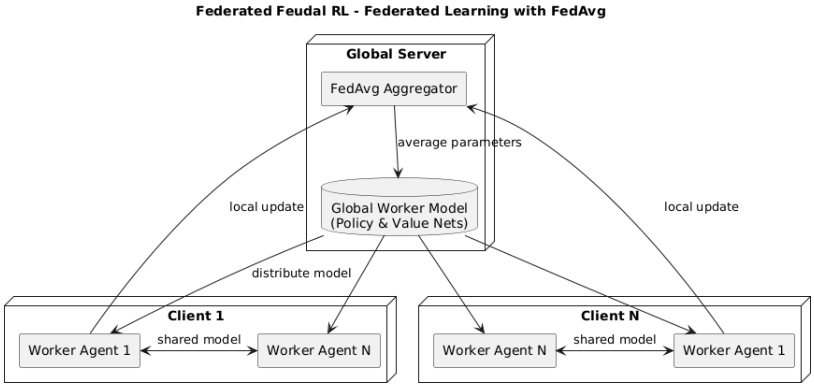


Figure 3.3: Global Worker Model. Author's own work.

The federated setup offers two critical advantages:

- **Data locality and privacy:** Raw environment trajectories and agent observations never leave the local environment. Only essencial model parameters are exchanged, making the system compatible with privacy-sensitive or resource-constrained deployments.
- **Scalability and heterogeneity support:** Clients may simulate varied local conditions, including differing initial positions and randomized targets. The combination of local updates and global aggregation enables the model to generalize across heterogeneous data distributions while preventing overfitting to any single client's environment.

- **Proximal Regularization (FedProx).** To promote stability during local training, we include a quadratic penalty that discourages large deviations from the global worker model during local optimization. This regularization term helps align updates across clients, mitigating the effects of stochastic variation introduced by randomized environment initializations (e.g., drone and target positions). While all clients operate in structurally

identical environments, such randomness can still cause divergence in local experiences. Importantly, only the worker agents participate in federated learning; the `ManagerAgent` remains centralized, preserving a separation between high-level planning and decentralized execution.

2.2 Key Design Decisions

- **Number of Clients and Workers per Client:**
  A configuration of 4 clients, each hosting 2 workers, was selected as a practical trade-off between communication overhead, statistical diversity, and system complexity. This setup ensures that there is enough inter-client variability to benefit from federated averaging, while keeping the synchronization costs and potential instability manageable.

  Choosing more than two workers per client would have required expanding the spatial and coordination complexity of the environment to prevent agent overlap, communication congestion, or reward interference. To preserve an interpretable and tractable multi-agent dynamic, the worker count was limited to two per client—sufficient to study coordination and exploration without overwhelming the system with combinatorial complexity.

  As for the number of clients, values between 2 and 4 were considered. A single client was ruled out due to the lack of inter-client variability, which leads to brittle convergence and poor generalization. Although 2 or 3 clients would have been viable, 4 was chosen to strengthen diversity and better emulate a federated scenario, while still avoiding extreme asynchrony or instability. Using more than 4 clients was deemed too ambitious due to the increased risk of performance degradation from noisy or unstable individual clients. See Fig. 3.4 from a 4 clients scenario. The specifics of the evaluation protocol and metrics are discussed in detail in Subsection 3.3.

  To illustrate the contrast, Fig. 3.5 shows the result of a single-client training run. Although this setup fails to generalize and maintains a low capture rate, it exhibits a stable learning trajectory without sudden performance collapses. The low capture rate with an increasing reward is explained later in this section as a failure to adquire the sparse reward signal, as visualized in Fig. 3.6.

  It is important to note that a single-client environment is not suitable for exploring the privacy-oriented federated setting. Moreover, while federated averaging can increase sensitivity to instabilities at individual clients, it is likely not the sole cause of the observed performance drops. Environmental stochasticity, network update delays, and initialization variance may also contribute to such dynamics.

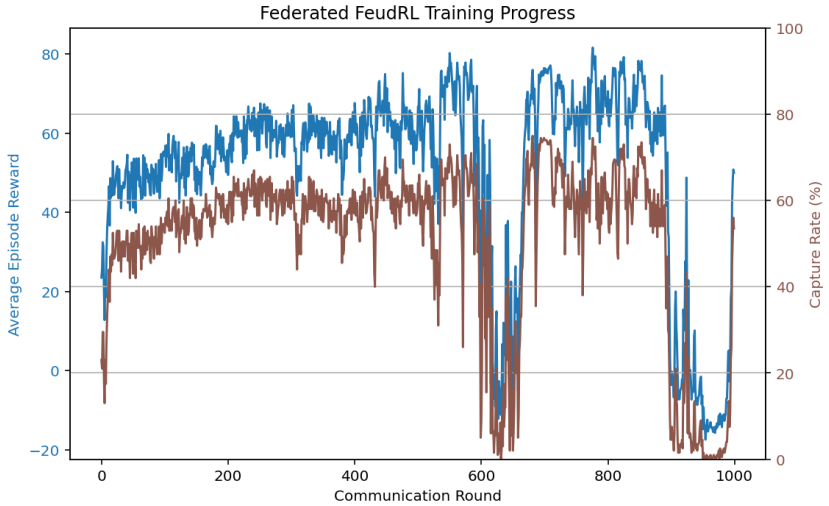


Figure 3.4: Sudden performance drop observed during training, potentially caused by an unstable client model.

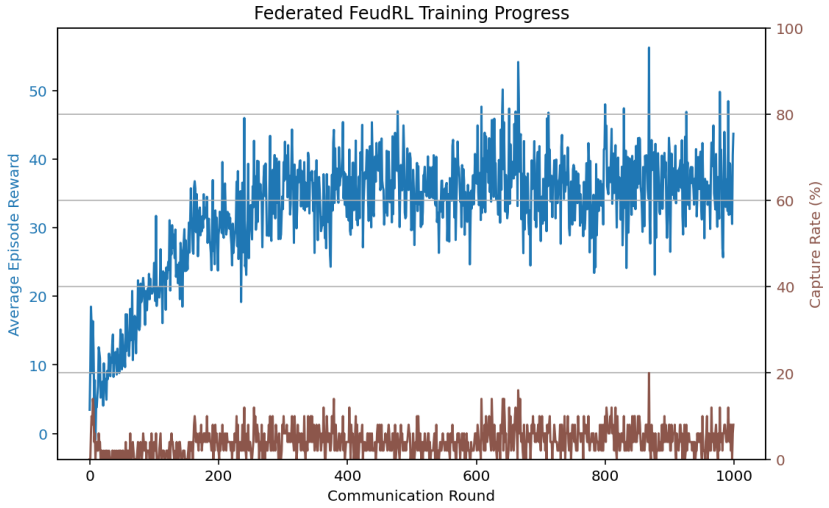


Figure 3.5: Single client training without drops.

- **Network Architectures:**

  Input and Output Dimensions: Both the manager and the workers use fully connected multilayer perceptrons (MLPs) with two hidden layers and ReLU activations. The input and output dimensions are structured as follows:

  – *Manager Policy Network:* The manager receives a global state vector in $\mathbb{R}^{3n+3+3}$, where n is the number of worker agents. This vector is constructed by concatenating the 3D positions of all n agents with the 3D position of the target and its 3D velocity vector. Importantly, this state is not directly observed by the manager, but rather constructed from partial observations shared by the workers. This design preserves the privacy-oriented nature of the federated setup, in which the manager does not

have direct access to the environment and instead relies on structured input from decentralized sources. The network then outputs the parameters (mean and standard deviation) of a Gaussian distribution over goals in $\mathbb{R}^{n\times 3}$ — one 3D goal vector per worker.

– *Worker Policy Network:* Receives a concatenation of the agent's local state and the goal vector from the manager, resulting in an input of size $\mathbb{R}^6$. It outputs the parameters of an action distribution in $\mathbb{R}^3$, used to sample the worker's continuous action.

Hidden Layer Size: A hidden size of 64 was selected for the manager, and 192 for the worker agent, based on Optuna-based Bayesian hyperparameter tuning. This size was found to offer an effective trade-off between model capacity and training stability. Larger sizes showed marginal gains at the cost of significantly higher computation and increased variance under client heterogeneity.

Critic Networks: Both the manager and the workers include separate value networks. This design decouples policy and value optimization, which improves stability and avoids conflicting gradient signals during actor-critic updates. Sharing parameters between actor and critic was deliberately avoided to reduce interference.

- **Reward Structure:**
  The reward function plays a central role in shaping agent behavior. It was designed to balance target proximity, collision avoidance, and strategic coordination. Specifically, each worker receives a shaped reward composed of:

  – **Positive reward for progress toward the target:** Measured as the reduction in distance to the target compared to the previous timestep. It is computed as the reduction in Euclidean distance to the target between successive timesteps:

  $$r^i_{\text{target}}(t) = d^i(t-1) - d^i(t)$$

  where $d^i(t) = \|\mathbf{p}_i(t) - \mathbf{p}_{\text{target}}(t)\|_2$ is the distance between agent i and the target at time t.

  – **Minimal encouragement for spacing from other drones:** A small weight is placed on the average distance to other agents to promote spatial diversity without undermining goal-seeking behavior. Formulated as:

  $$r^i_{\text{spread}}(t) = \frac{1}{N-1}\sum_{j\neq i} \|\mathbf{p}_i(t) - \mathbf{p}_j(t)\|_2$$

This term discourages clustering, reducing potential for collisions while improving coverage.

These components are combined linearly, with target proximity weighted at 0.9995 and inter-agent spacing at 0.0005, emphasizing task completion over group spread.

- **Negative penalties:**
  - -10 for collisions between agents.
  - -20 for collisions with the ground (i.e., flying too low).
- **Sparse capture reward:** A reward of $+100$ is given if an agent enters the proximity radius of the target.

The reward structure formulation can be summarized as:

$$r^i(t) = \begin{cases} -10 & \text{if collision with another drone} \\ -20 & \text{if collision with the ground} \\ +100 & \text{if target is caught} \\ 0.9995 \cdot r^i_{\text{target}}(t) + 0.0005 \cdot r^i_{\text{spread}}(t) & \text{otherwise} \end{cases}$$

This reward design encourages agents to efficiently surround and intercept the moving target while avoiding unsafe or inefficient behaviors. The balance was empirically tuned to provide sufficient learning signals without destabilizing training or encouraging undesirable emergent strategies (e.g., clustering or over-separation).

Given that each episode is capped at 50 steps (see upcoming Subsection 3.4 for details) and the movement per timestep is limited to 0.2 units, agents can accumulate small shaped rewards for gradually approaching the target. The shaped reward component—weighted at 0.9995—typically yields between 10 and 30 points per episode when agents act efficiently. In contrast, the inter-agent spacing component (weighted at 0.0005) contributes only marginally to the total reward.

When the target is successfully intercepted—which grants a one-time reward of $+100$—the total per-agent reward in an ideal episode can reach approximately 110 to 130 points. See Fig. 3.6 for a theoretical representation of such training trajectory,where the sparse reward signal is first captured in round 6. This balance encourages agents not only to approach the target, but to do so efficiently. Because the capture reward dominates over shaped proximity rewards, it discourages suboptimal or hesitant chasing behavior and promotes decisive interception strategies.

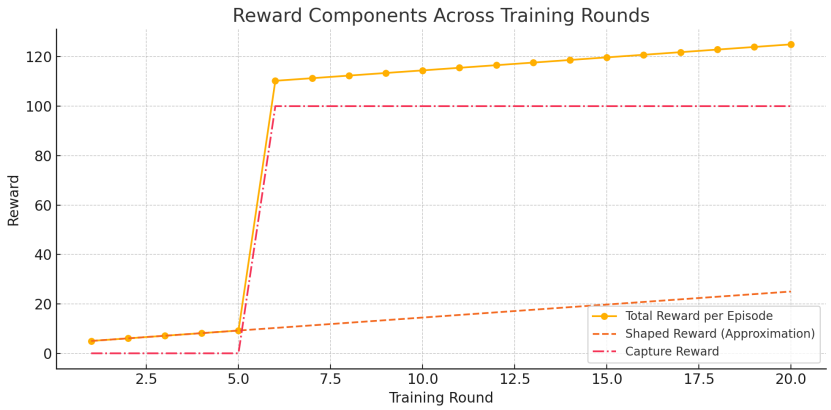


Figure 3.6: Reward progression in a theoretical scenario.

- **Hyperparameter Tuning:**
  The goal of the tuning process was to maximize the average episode reward during the final communication rounds, providing a robust estimate of the learned policy's quality in a federated and hierarchical context. Using Optuna trialing as well, we tuned the following hyperparameters:

  - **Learning Rate:** Controls the step size in gradient descent updates. Separate learning rates were tuned for the manager and worker agents to balance convergence speed and training stability.
  - **Entropy Coefficient:** Scales the entropy term in the policy loss, encouraging exploration by discouraging overly deterministic policies. Higher values promote more diverse action sampling.
  - **Discount Factor γ:** Determines how future rewards are weighted relative to immediate rewards. A value close to 1 emphasizes long-term planning, whereas lower values prioritize short-term gains.
  - **FedProx Coefficient μ:** Adds a proximal regularization term during local worker updates to penalize divergence from the current global model. This helps stabilize federated learning by reducing inconsistencies introduced by local randomness in the environment.
  - **GAE Parameter λ:** Governs the bias–variance trade-off in Generalized Advantage Estimation. Lower values introduce more bias but reduce variance; higher values provide more accurate long-term credit assignment at the cost of increased variance.

  The final configuration resulting from the Optuna-guided tuning process, along with all subsequent tuning results, is summarized in Table 5.1. These values represent the best

trade-off observed between learning efficiency, model stability, and generalization performance in the federated and hierarchical setting. Notably, the manager and worker agents were assigned different learning rates to reflect their distinct optimization dynamics, and a relatively low FedProx coefficient was selected to allow sufficient local adaptation while still preventing client divergence.

- **Early Stopping Criterion:**
  Threshold: 95% average capture rate over 5 consecutive communication rounds.
  Reasoning: Prevent overtraining and reduce unnecessary computation given the sustainability implications of redundant training.

3 Methodology

3.1 Logging, Model Saving, and Evaluation Utilities

A full suite of utilities was implemented to support logging, model saving, and evaluation throughout development. These tools were essential for reproducibility, diagnostics, and result interpretation across experiments. A detailed description of this infrastructure, including hyperparameter logging, model serialization, evaluation procedures, and training visualization tools, can be found in Annex 2.

3.2 Training Experience

3.2.1 Setup specifications

All experiments were conducted on a local workstation with the following hardware and software configuration:

- **Operating System:** Microsoft Windows 11 Pro (Build 26100)
- **CPU:** Intel Core i9-11900K (8 cores, 16 threads, base clock 3.50 GHz)
- **Motherboard:** ASUS ROG STRIX Z590-F
- **RAM:** 32 GB DDR4
- **GPU:** NVIDIA GeForce RTX 3090
- **Python:** 3.11 (64-bit)
- **PyTorch:** 2.5.1 with CUDA 12.1 support
- **PettingZoo:** 1.24.3

3.2.2 Initialization

The initial positions of the n drones and the target are randomized using uniform sampling, constrained by a minimum separation radius. This avoids trivial collisions or overlapping configurations, and ensures diverse starting scenarios for learning.

3.2.3 Local Episodes

Each client runs E feudal episodes per communication round. During each episode:

- At every k steps, the manager samples a new per-agent goal vector from a Gaussian distribution parameterized by its policy network.
- Worker agents select actions based on their local state concatenated with the current manager goal ($s_i \oplus g_i$).
- Actor-critic updates are applied using Generalized Advantage Estimation (GAE) with coefficient λ.
- If a global worker snapshot is available, a FedProx regularization term is added to the worker loss to discourage excessive deviation.

3.2.4 Aggregation

After E local episodes, worker network parameters are aggregated across all clients using Federated Averaging (FedAvg). This produces a global worker model that is redistributed before the next round.

3.2.5 Gradient Clipping:

During training, gradients for both policy and value networks are clipped to a maximum norm of 5.0. This regularization technique is especially important in federated reinforcement learning, where client models may undergo large local updates between synchronization steps. Without clipping, such updates can result in:

- Instabilities caused by excessively large parameter changes.
- Divergence due to the compounding effect of high entropy exploration in early training.
- Slower convergence or erratic performance due to noisy gradients from limited batch sizes per client.

Empirically, gradient clipping was found to significantly reduce variance across training rounds and helped ensure convergence even when using relatively aggressive learning rates or early-stage hyperparameter values.

3.2.6 Learning-Rate Scheduling

Learning rates for both the manager and workers are decayed every 100 communication rounds using step scheduling (see Algorithm 1). This helps stabilize long-term convergence by gradually reducing the magnitude of updates as training progresses. Given that each communication round consists of many local episodes, decaying the learning rate more frequently (e.g., every few episodes) would result in overly aggressive attenuation. Such premature decay would bias the learning process toward exploitation too early, reducing the agents' ability to explore diverse strategies and increasing the risk of convergence to suboptimal local minima. By scheduling decay at the level of communication rounds rather than episodes, sufficient learning capacity is preserved during local adaptation, while the global learning dynamics remain stable and directed toward optimal convergence over time.

3.3 Evaluation

Throughout training, multiple evaluation metrics are tracked and recorded to monitor learning progress and enable cross-run comparisons. These include:

- **Per-Round Metrics:** For each communication round, we log:
  - *Average Reward:* The mean per-agent cumulative reward across all episodes and clients. This includes shaped rewards from progress toward the target, penalties for collisions, and the sparse +100 capture reward.
  - *Capture Rate:* The percentage of episodes in which at least one worker successfully intercepted the target (i.e., entered the proximity radius defined by `CLOSENESS_THRESHOLD` in the environment).
  - *Average Episode Length:* The mean number of steps executed per episode before termination or truncation. This helps assess both efficiency and episode-level dynamics.
- **Visualization:** Training curves are generated to visualize trends over time. Specifically:
  - Reward curves reveal whether agents are learning effective policies.
  - Capture rate curves indicate task success frequency and convergence stability.
  - Episode length plots reveal behavioral trends such as hesitancy or overly aggressive strategies.
- **Final Test Evaluation:** After training, a set of 100 held-out evaluation episodes is conducted using the final global models. These episodes are run in the "human" render mode to support visual inspection of agent behavior in real-time.

- Each test episode uses a new randomized initialization of drone and target positions.
- Only the first client's worker model is used, as all client models are synchronized post-aggregation.
- The success rate over these episodes—measured as the percentage of runs where the target is captured—serves as the final generalization score.

Together, these evaluation tools offer a comprehensive picture of training progress, convergence quality, and generalization potential in unseen scenarios. They also aid in diagnosing potential failure modes such as reward hacking, premature convergence, or overfitting to local environment dynamics.

3.4 Empirical Design Decisions

3.4.1 Episodes per Communication Round (E):

To better understand the trade-off between local adaptation and global synchronization, we conducted four controlled experiments where the product of communication rounds and local episodes was fixed at 5000 total episodes. The values of E tested were $\{100, 50, 20, 10\}$, corresponding respectively to 50, 100, 250 and 500 communication rounds, which are showcased in the Figs. 3.7, 3.8, 3.9 and 3.10.

Results showed a strong performance gradient. Higher values of E (e.g., $E = 100$) allowed for deeper local policy refinement and occasionally produced robust results. However, the reduced synchronization frequency increased the risk of local model drift—particularly in stochastic environments—resulting in inconsistent convergence across different runs. Conversely, low values such as $E = 10$ led to more frequent updates but insufficient local learning, yielding unstable training curves and poor overall performance.

This behavior highlights a key risk in federated reinforcement learning: when E is too small, clients may fail to accumulate enough experience to generate meaningful gradients, leading to underfitting and noise-dominated updates. When E is too large, the resulting updates—though better informed—can deviate too far from the global model, weakening the effect of aggregation and leading to inconsistent global behavior.

The configuration with 50 episodes per communication round (Fig. 3.8) demonstrated the most favorable balance: it consistently yielded high average rewards and capture rates across runs, maintained stable convergence, and had slightly better runtime performance. These characteristics made it a practical default choice in the remainder of this work.

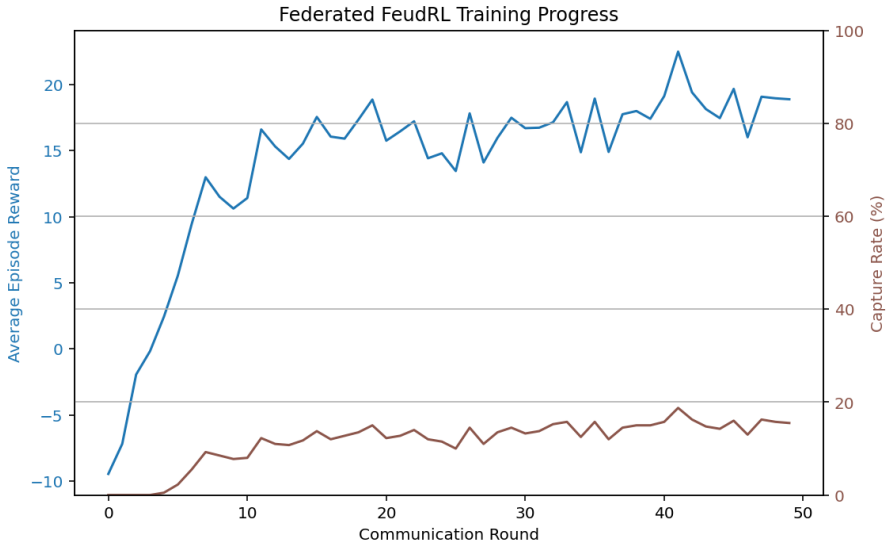


Figure 3.7: Setup with 50 communication rounds and 100 episodes each.

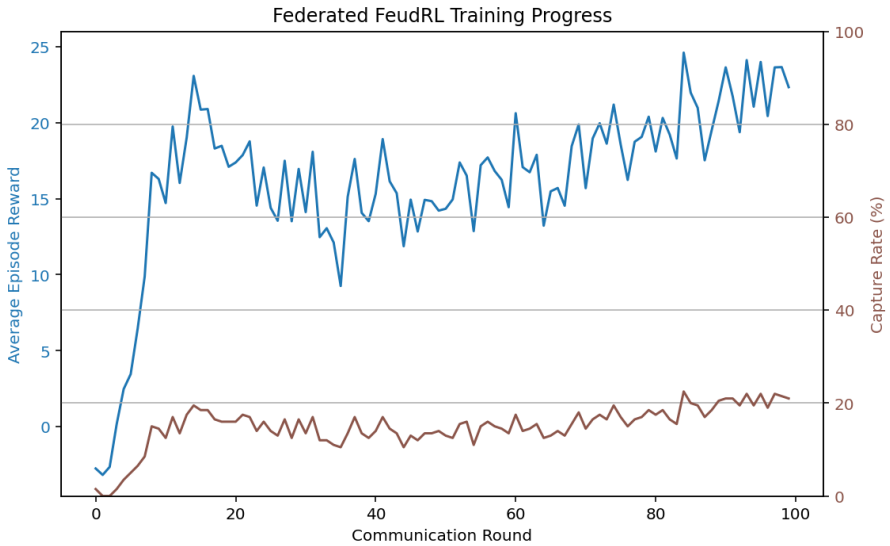


Figure 3.8: Setup with 100 communication rounds and 50 episodes each.

These results are illustrative of general trends, but each corresponds to a single run and is therefore not statistically representative on its own. During broader experimentation, we observed that high-E configurations (e.g., $E = 100$) occasionally achieved strong results but were less stable and more sensitive to randomness and initialization variance. In contrast, the $E = 50$ configuration exhibited consistently better convergence across multiple seeds and generally faster execution, making it the most robust setup for practical use.

In summary, configurations with very low E suffer from instability and ineffective updates, while very high E leads to greater client divergence and reduced generalization. Among the tested options, $E = 50$ provided the best trade-off—enabling meaningful local learning without

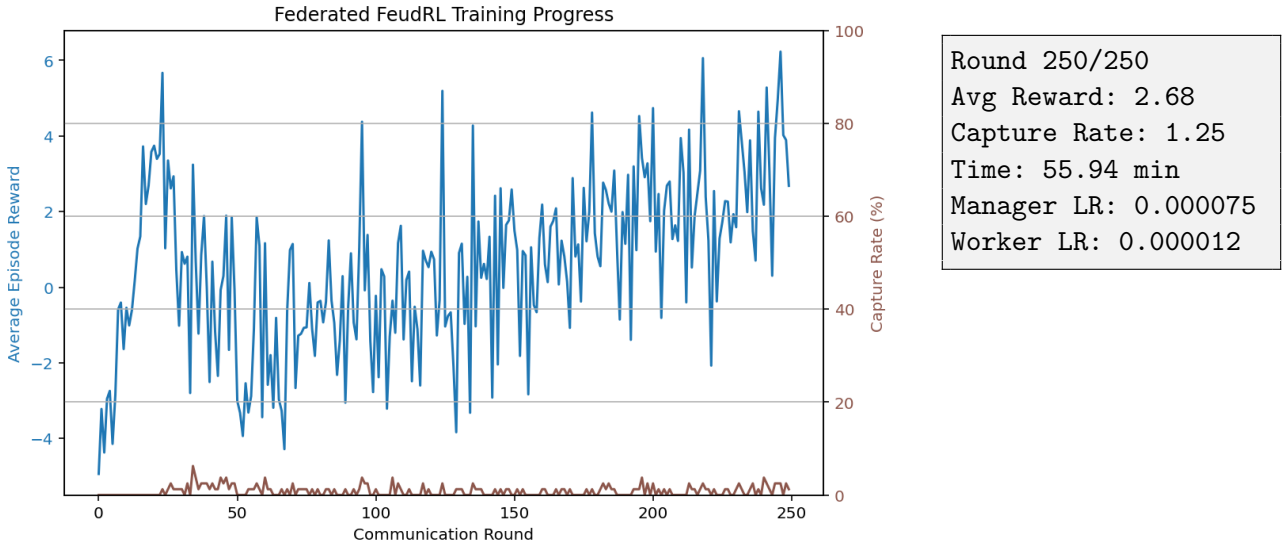


Figure 3.9: Setup with 250 communication rounds and 20 episodes each.

undermining global coordination. This empirical insight guided our choice of $E = 50$ as the default throughout the rest of this work.

3.4.2 Steps per Episode:

Multiple configurations were evaluated for the maximum number of steps per episode, ranging from 10 to 200 (See Figs. 3.11, 3.12 and 3.13). Reaching the steps limit results in an automatic termination of the training episode and the rewards for the run would be locked to the accumulated reward to that point. There is no explicit penalization for reaching the termination stage. Shorter episodes (see Figs. 3.11) imposed overly strict temporal constraints. Agents often lacked sufficient time to explore the environment, let alone coordinate toward capturing the target, terminating before meaningful behaviors could emerge. Conversely, longer episodes (e.g., 200 steps) extended training times and diluted the impact of the 100-point capture reward (see Figs. 3.13). In such cases, the shaped distance-based rewards accumulated over many steps could overwhelm the intended learning signal.

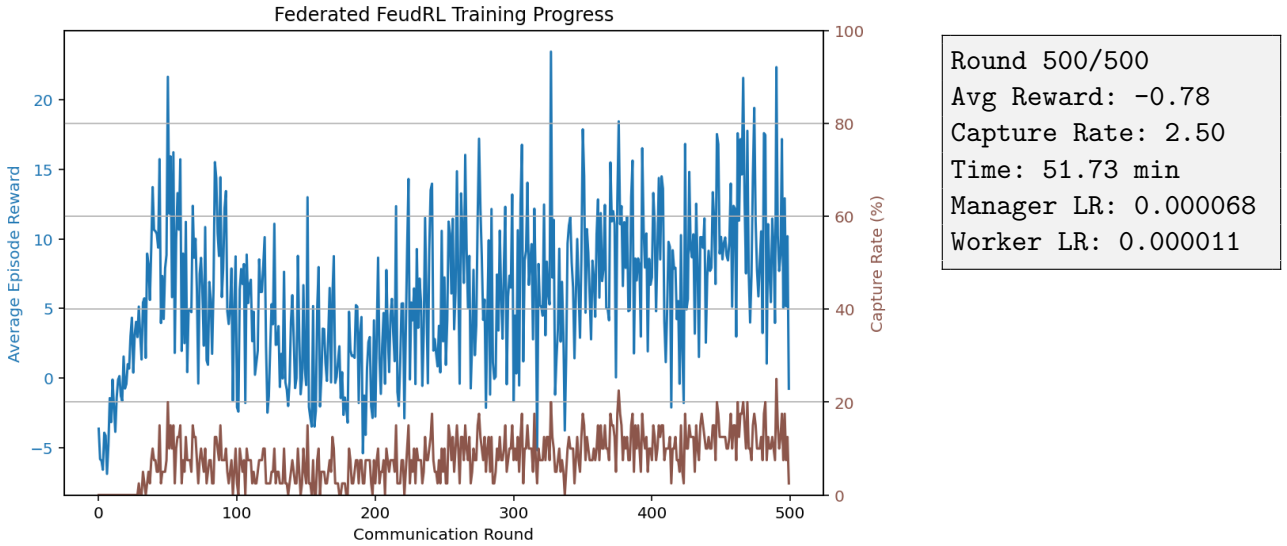


Figure 3.10: Setup with 500 communication rounds and 10 episodes each.

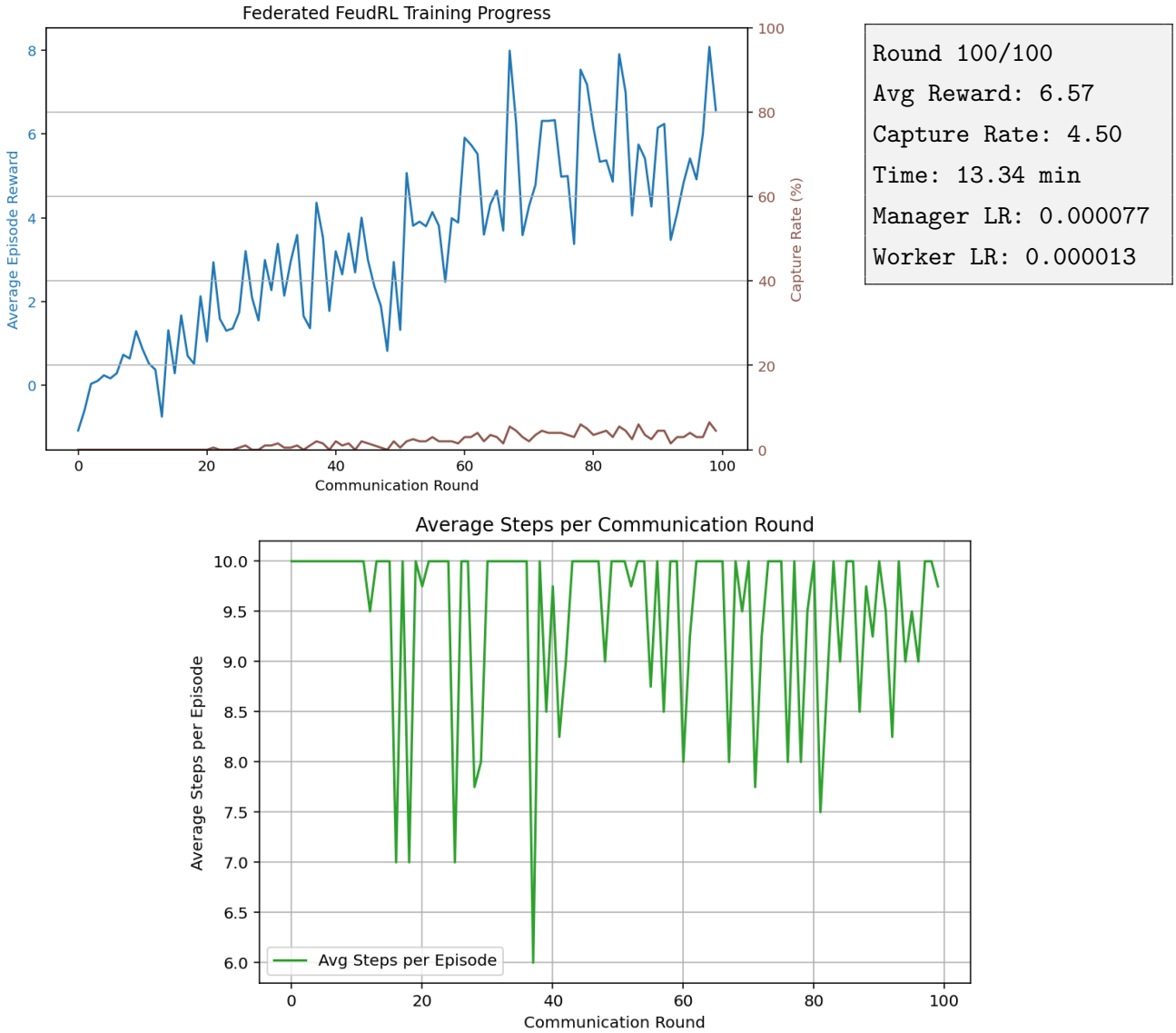
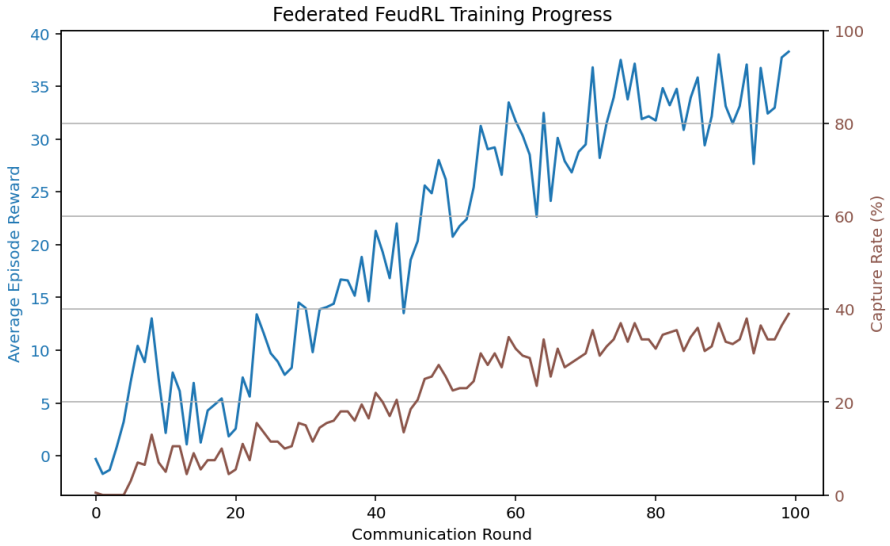


Figure 3.11: Steps progression for a setup with 10 steps per episode.



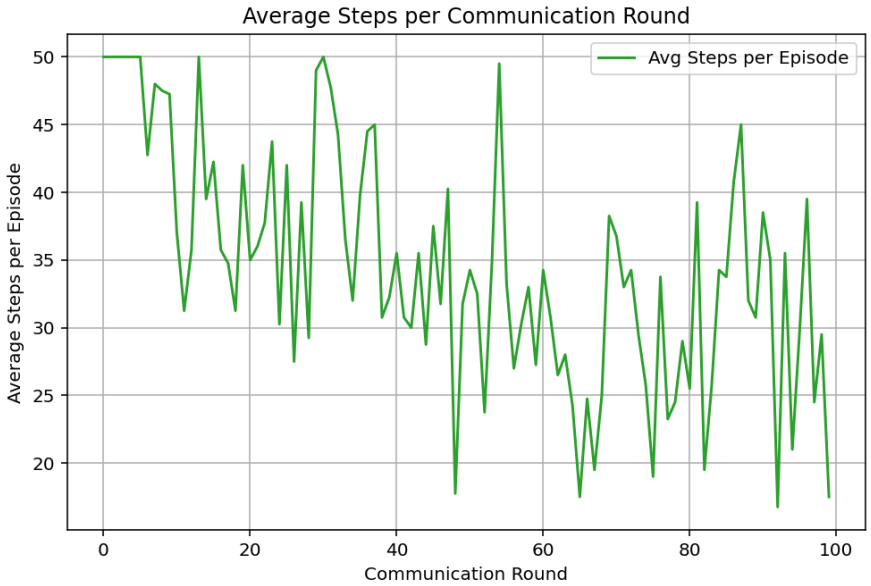


Figure 3.12: Steps progression for a setup with 50 steps per episode.

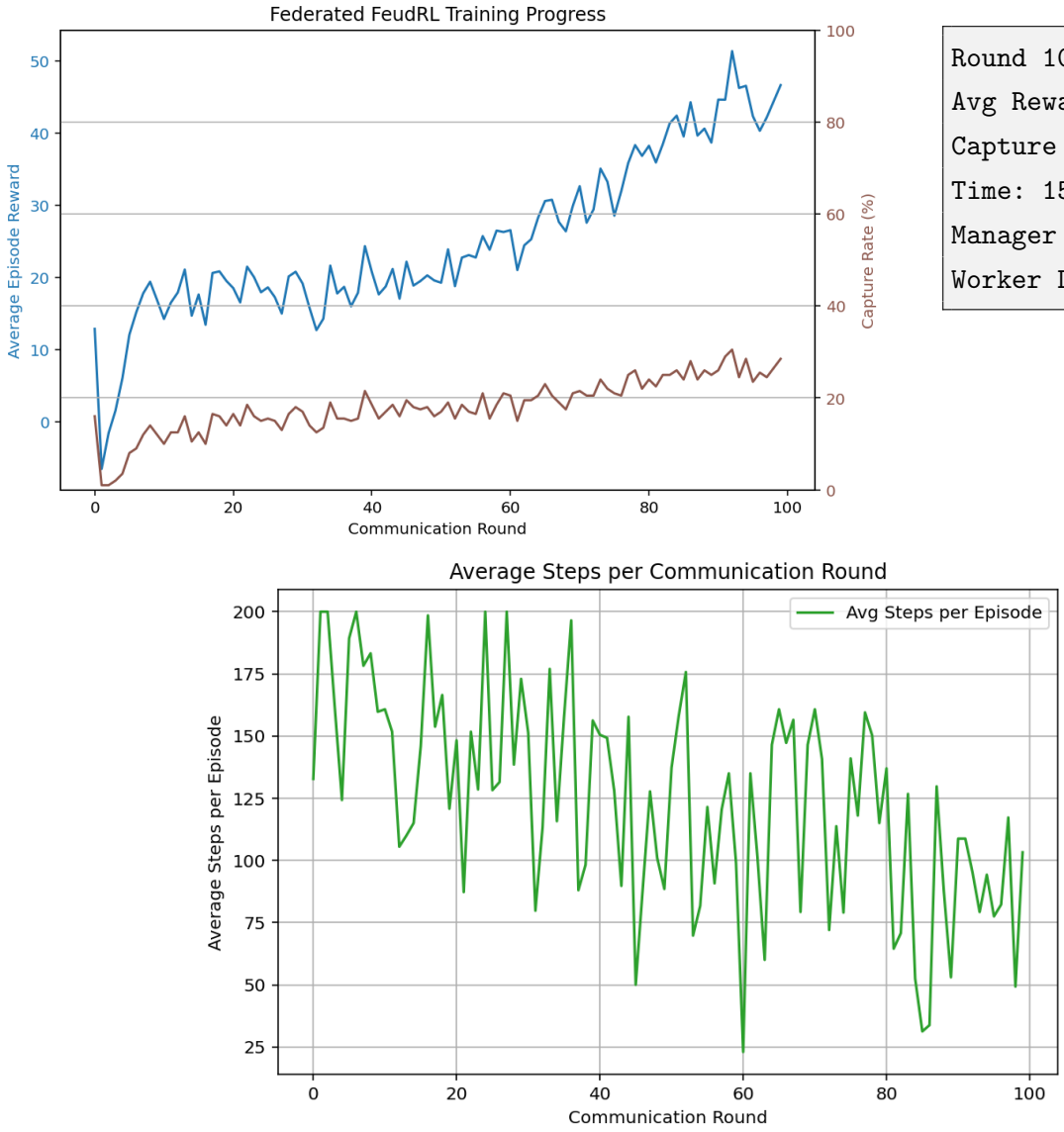


Figure 3.13: Steps progression for a setup with 200 steps per episode.

We can clearly observe between Fig 3.12 and Fig 3.13, that when episodes allow too many steps (as a contrast to Fig 3.11), agents tend to exploit the extended time budget rather than optimize for efficient target capture. This behavior suggests a reduced incentive to find the shortest or most effective path to success. Therefore, selecting an appropriate episode length is crucial: it must reflect the environment's spatial and temporal constraints, as well as its inherent randomness, to guide agents toward meaningful and efficient policies.

It is also important to note that these findings are tied to the specific reward structure used in our implementation. For instance, introducing a small negative reward per step could encourage agents to minimize path length—but at the same time, it may introduce noise that

competes with the shaped approximation reward, potentially destabilizing learning. A different reward design—such as a higher weight on spacing penalties, or a time-dependent decay on the capture reward—might lead to different optimal episode lengths and learning dynamics. Thus, the conclusions drawn here are valid within the context of our current reward shaping strategy, and would need to be re-evaluated under alternative formulations.

In summary, while longer episodes increased raw episode rewards, they did not translate proportionally to improved task completion (capture), especially accounting for the excess of computational requirements. Short episodes restricted learning opportunities. The 50-step configuration (see Figs. 3.12) balanced exploration, reward informativeness, and training time, emerging as the most practical and effective setting for downstream evaluations.

3.4.3 Empirical analysis of GAE and FedProx combination:

To understand the role of Generalized Advantage Estimation (GAE) and FedProx regularization in federated feudal reinforcement learning, we conducted a controlled comparison involving four setups:

- Using both GAE and FedProx (baseline),
- Disabling GAE,
- Disabling FedProx,
- Disabling both GAE and FedProx.

Disabling FedProx was achieved by setting $\mu = 0$, while disabling GAE corresponded to setting $\lambda = 1$ in the GAE formulation. These adjustments modify only the mathematical properties of optimization and advantage estimation without significantly affecting computational time. The detailed implementation and analysis of the combined use of GAE and FedProx, including algorithmic formulation and computational considerations, are provided in Annex 4.

We used a consistent setup of 100 communication rounds across all trials. The configuration using both GAE and FedProx is shown in Fig. 3.12, previously introduced in the steps-per-episode analysis.

Baseline: GAE and FedProx Enabled

(Previously shown) Fig. 3.12

```
Round 100/100 - Avg Reward: 38.30 - Capture Rate: 39.00 | Time: 43.41 min | Manager
    ↪ LR: 0.000077 | Worker LR: 0.000013
```

Disabling GAE

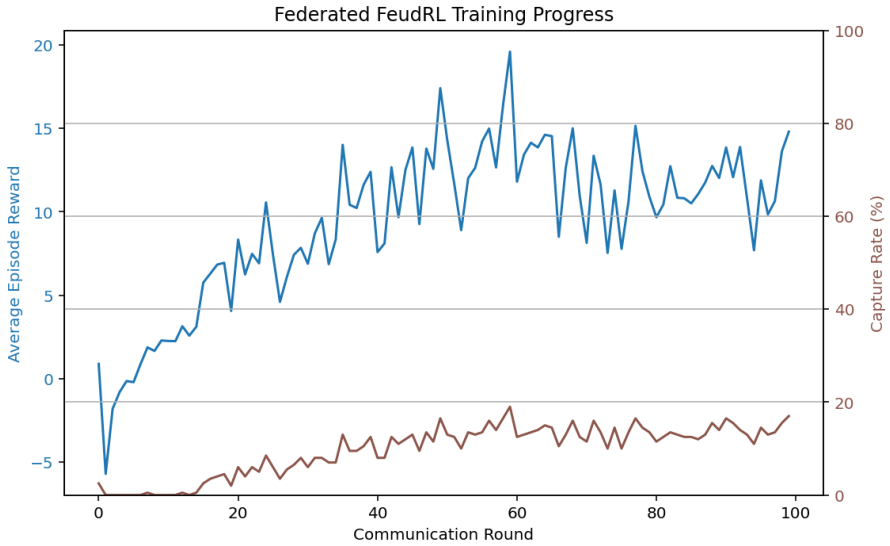


Figure 3.14: Reward curve for a setup without GAE (only FedProx active).

Disabling GAE (while keeping FedProx) significantly degraded both reward and capture rate. Advantage estimation without temporal smoothing increased variance and destabilized training dynamics, leading to noisier optimization and less consistent progress.

Disabling FedProx

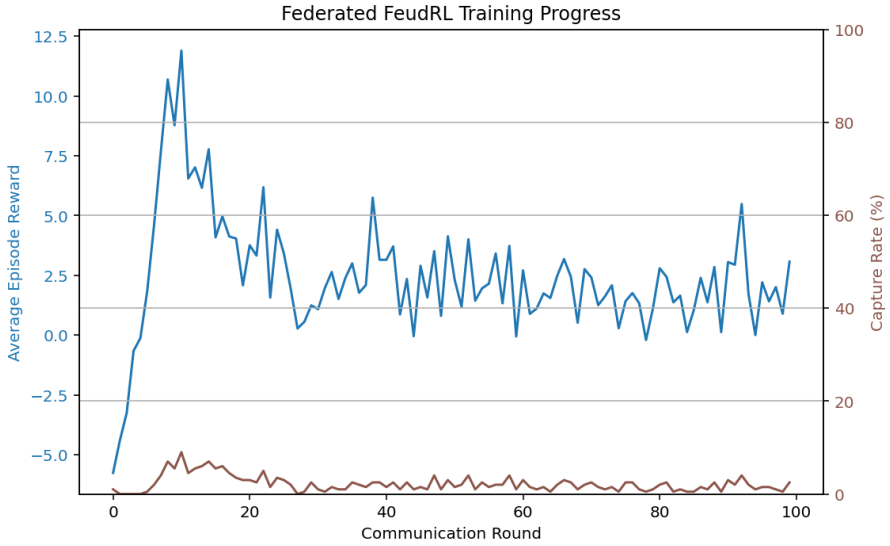


Figure 3.15: Reward curve for a setup without FedProx Averaging (only GAE active).

Disabling FedProx alone (while keeping GAE) also worsened convergence, primarily by inducing more erratic updates across clients—likely due to increased local model drift that FedProx would normally penalize. Of particular note, the phenomenon of *unlearning* impacts becomes more evident compared to previous observations (see Fig. 3.4), as the model appears less capable of recovering once destabilized without the regularization effect of FedProx.

Disabling Both GAE and FedProx

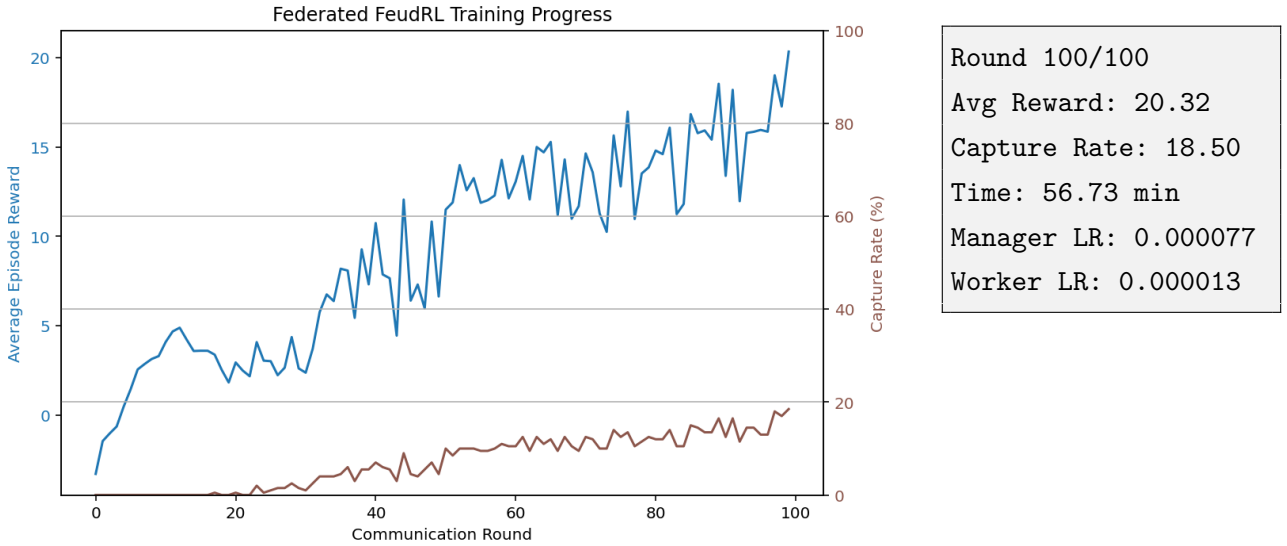


Figure 3.16: Reward curve for a setup without either GAE or FedProx Averaging.

Interestingly, disabling both GAE and FedProx occasionally produced higher average rewards than disabling only one; however, training was markedly less stable, suggesting that missing both regularization and advantage smoothing leads to unreliable optimization behavior. However, the training curves were less stable than the baseline, suggesting that missing both regularization and advantage smoothing can lead to overfitting or unstable optimization patterns despite occasional reward spikes.

3.4.4 Findings on GAE and FedProx

Configuration	Avg. Reward	Capture Rate (%)
GAE + FedProx (Baseline)	38.30	39.00
No GAE, FedProx only	14.81	17.00
GAE only, No FedProx	3.07	2.50
No GAE, No FedProx	20.32	18.50

Table 3.1: Impact of GAE and FedProx on performance.

Overall, while individual runs may vary due to stochasticity, the general pattern strongly favors the combined use of GAE and FedProx to promote smoother convergence, stabilize local updates, and improve robustness during federated reinforcement learning.

Based on these results, we trained was performed using FedAvg together with GAE and FedProx. Figure 3.17 shows that, after 1 000 rounds, the FedAvg+GAE+FedProx configuration reached an average reward of 78.09 with a 46% capture rate.

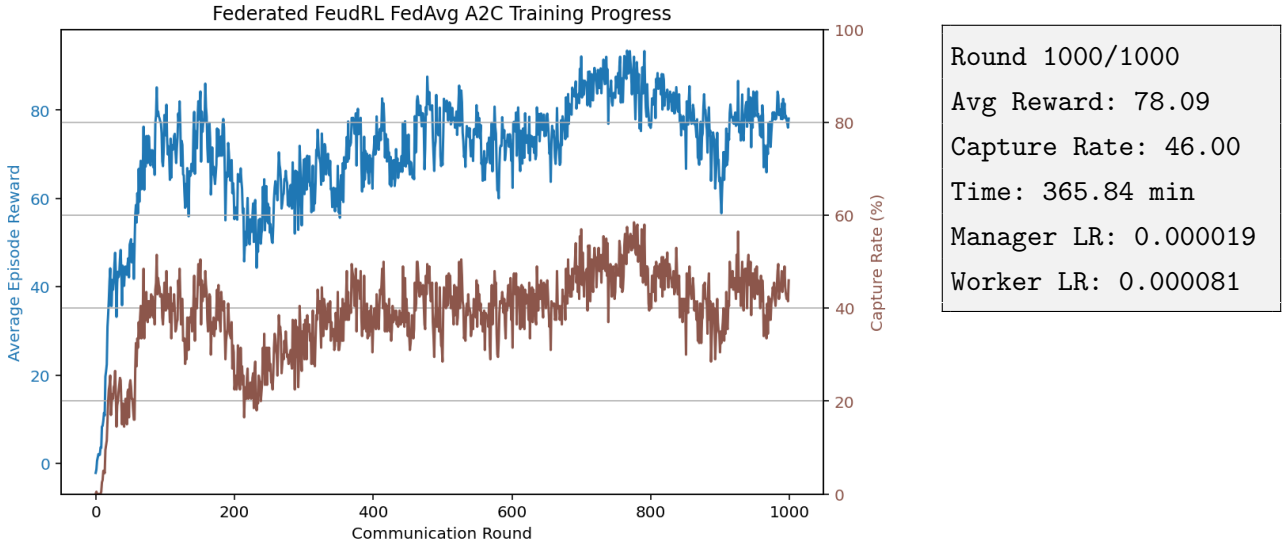


Figure 3.17: Reward curve for a setup with FedAvg.

3.5 Alternative Approaches

Having established FedAvg with GAE and FedProx as our baseline, we next evaluated three variants to see if they could further improve performance.

3.5.1 Hierarchical MAPPO under FedAvg & FedProx

Based on our A2C experiments and the challenges of continuous, multi-agent, sparse-reward Catch, we wanted to explore a more complex alternative to the basic actor-critic by using a hierarchical, multi-agent Proximal Policy Optimization (MAPPO) backbone. This leverages three key ideas:

- **Why PPO & MAPPO?**
  - *Clipped surrogate loss:* PPO's clipped objective

$$\mathcal{L}_{\text{clip}}(\theta) = \mathbb{E}\Big[\min\big(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\,\hat{A}_t\big)\Big]$$

    strictly bounds each policy update, taming the high-variance "blowups" typical of on-policy AC when combined with FedAvg and FedProx.
  - *Sample efficiency:* By re-using each batch of trajectories for multiple epochs (4–10), PPO dramatically improves learning speed when clients collect only a handful of episodes between communication steps.

  - *Proven multi-agent performance:* Centralized-critic, decentralized-actor PPO (MAPPO) is often state-of-the-art on cooperative continuous benchmarks (e.g. StarCraft micromanagement, multi-robot).

- **Integration into our federated feudal setup**
  1. *Manager and worker as separate PPO agents:*
     - *Manager:* treats the global state (including target velocity) as observation and emits sub-goals as actions, trained with PPO's clipped actor and a value network.
     - *Workers:* each worker sees (local state + goal), runs its own PPO loop; FedAvg and a FedProx term $\frac{\mu}{2}\|\theta - \theta_{\text{global}}\|^2$ wrap each client's optimizer step.
  2. *Centralized critics:* During training, both manager- and worker-critics see full global state (for stronger advantage estimates) even though at execution each worker only uses local+goal.
  3. *Federated schedule:*
     - Each communication round: clients collect K episodes, perform N_{ppo} PPO epochs *locally*, then FedAvg-average weights and continue.
     - FedProx penalty and PPO clipping together ensure local stability and global cohesion.
- **Expected benefits:**
  - *Stability:* PPO's trust-region–like clipping guards against large updates that destroy federated consensus.
  - *Sample reuse:* critical when each client collects few episodes.
  - *Seamless FedProx integration:* adding the $\|\theta - \theta_{\text{global}}\|^2$ term to the PPO loss is straightforward.

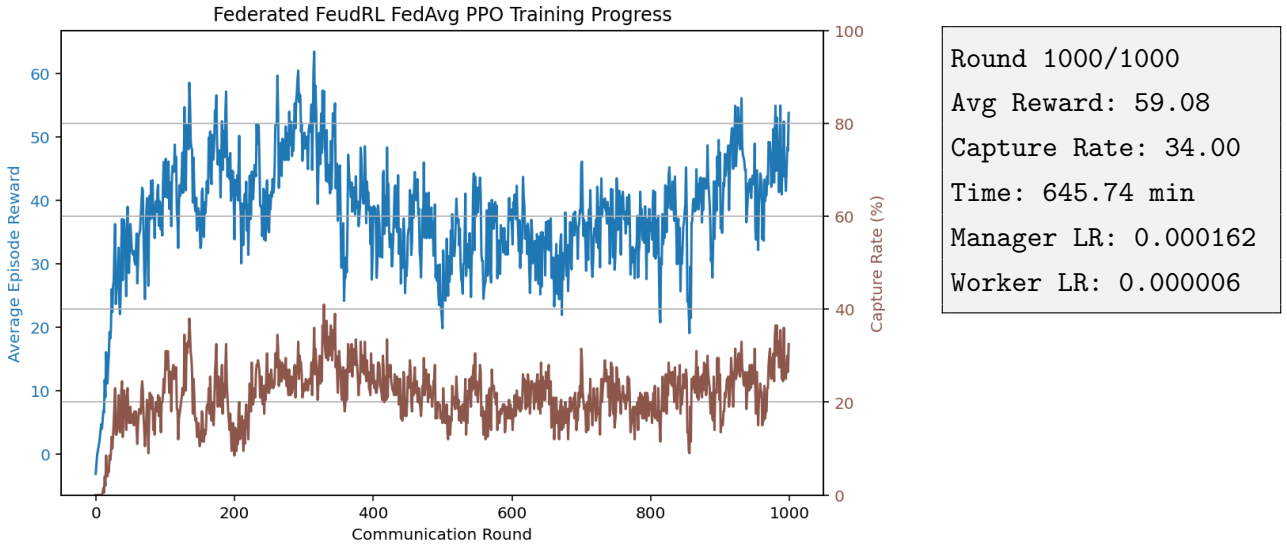


Figure 3.18: Reward curve for a setup with FedAvg using PPO.

The hierarchical MAPPO approach under FedAvg & FedProx did not outperform the simpler FedAvg A2C baseline. As shown in Fig. 3.18, the FedAvg A2C configuration achieved higher rewards and capture rates overall, despite MAPPO's theoretically stronger backbone.

3.5.2 Population Based Training under FedAvg & FedProx

In response to the client degradation behavior observed in Fig. 3.4, where weaker clients in Federated Averaging (FedAvg) pulled down the performance of stronger participants, we conducted an additional evaluation using a Population-Based Training (PBT) approach. This setup allowed us to isolate and compare the global performance when selection pressure is applied to favor higher-performing clients.

- **PBT Aggregation:** Instead of standard averaging, each communication round selected a subset of top-performing clients based on recent episode rewards. Only these clients contributed updates to the global model.
- **GAE and FedProx Retention:** The PBT variant retained both Generalized Advantage Estimation (GAE) and FedProx regularization as fixed components, given their proven contribution to stability and convergence in prior evaluations (see Fig.3.14, Fig.3.15). While PBT introduces selection pressure and adaptive exploration across client configurations, it does not inherently mitigate issues arising from local gradient noise (addressed by GAE) or inter-client drift (mitigated by FedProx). GAE continues to provide low-variance advantage estimates, which are essential for learning from sparse or shaped rewards. Similarly, FedProx remains relevant even in a PBT context, as client heterogeneity and asynchronous progress persist; without it, selected clients might still overfit their local environments or diverge from global consensus. Thus, while PBT optimizes

which clients contribute, FedProx and GAE ensure that how those clients learn remains stable and generalizable.

- **Hyperparameter Re-evaluation:** Prior to the PBT experiments, we re-optimized key hyperparameters—such as learning rates and the FedProx regularization strength (μ)—using the Optuna optimization framework.

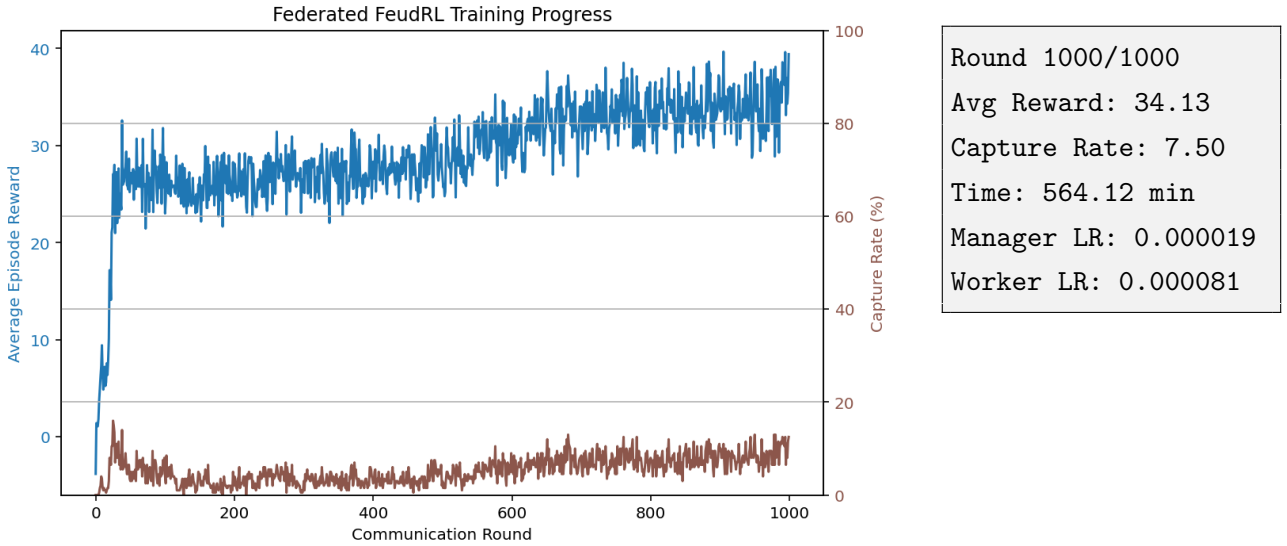


Figure 3.19: Reward curve for a setup with PBT.

The use of Population-Based Training (PBT) in the federated A2C setup did not yield superior performance compared to standard Federated Averaging (FedAvg). As illustrated in Fig. 3.19, while the reward curve exhibited more stable learning dynamics over communication rounds, both the average episode reward and capture rate remained consistently low throughout training.

3.5.3 Basic Population Based Training

After observing the underwhelming performance of PBT with GAE and FedProx retention, we explored a simplified variant that completely removed these stabilization mechanisms—Generalized Advantage Estimation (GAE), FedProx regularization, and Gradient clipping. The rationale was to assess whether PBT's adaptive client selection and inherent exploration could be sufficient to guide learning in the absence of added bias or regularization.

This basic PBT setup preserved the same overall federated structure and communication schedule, but client training with no regularization toward the global model (i.e., FedProx was omitted) and relied solely on empirical returns (rather than GAE) for policy updates.

Key differences compared to previous PBT:

- **No GAE:** Advantage estimates were computed using raw discounted returns, which tendo to be are more volatile but unbiased.
- **No FedProx:** Clients trained purely locally with no penalty toward global model parameters.
- **No Gradient Clipping:** Local policy gradients were left unconstrained, potentially allowing for higher-variance updates.

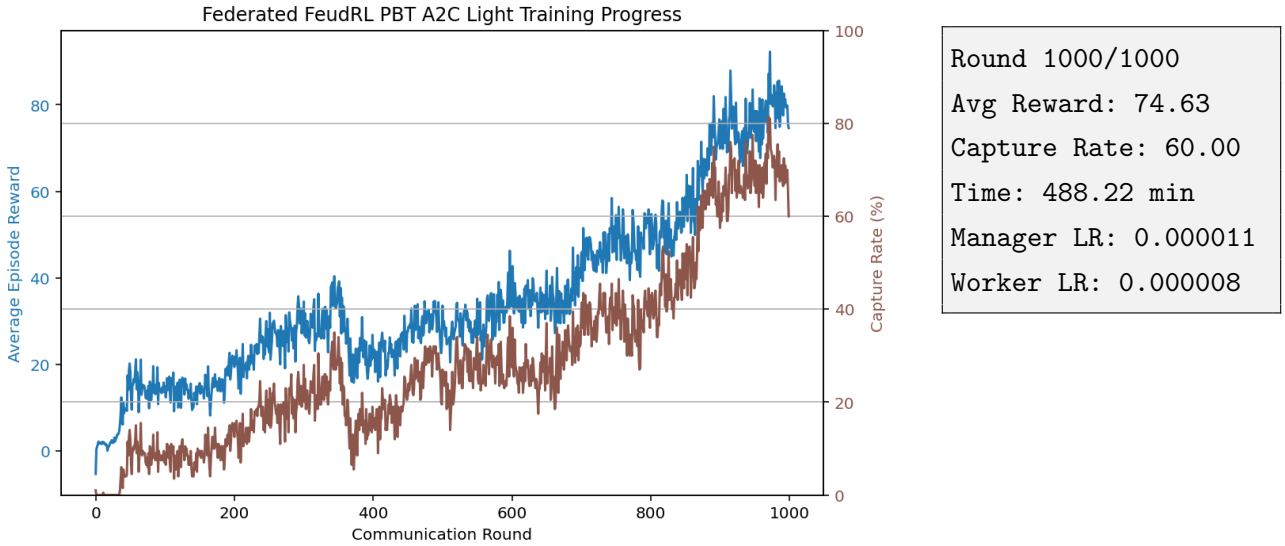


Figure 3.20: Reward curve for a basic setup with PBT.

Surprisingly, this stripped-down configuration produced one of the strongest outcomes across all evaluated methods. As shown in Fig. 3.20, the final average reward reached 74.63 with a capture rate of 60.00%—surpassing even the FedAvg with GAE and FedProx baseline.

These outcomes suggest that, in some settings, the added complexity of stability-oriented components may not always be necessary—especially when PBT is used to continually adapt client participation. That said, further experiments are needed to understand the generalizability of this result across tasks and environments.

Chapter 4

Results

1 Training Configurations Performance

This section summarizes the empirical outcomes of our federated feudal reinforcement learning experiments, as visualized in Fig. 4.1. We compare the performance of four configurations: FedAvg A2C, PBT A2C, FedAvg PPO (MAPPO) and Basic PBT A2C. To ensure robustness and reduce the influence of stochastic variance, we conducted five independent training runs for each configuration. From these, we selected the best-performing run based on final average reward and capture rate metrics for comparison in the figures and discussion that follow.

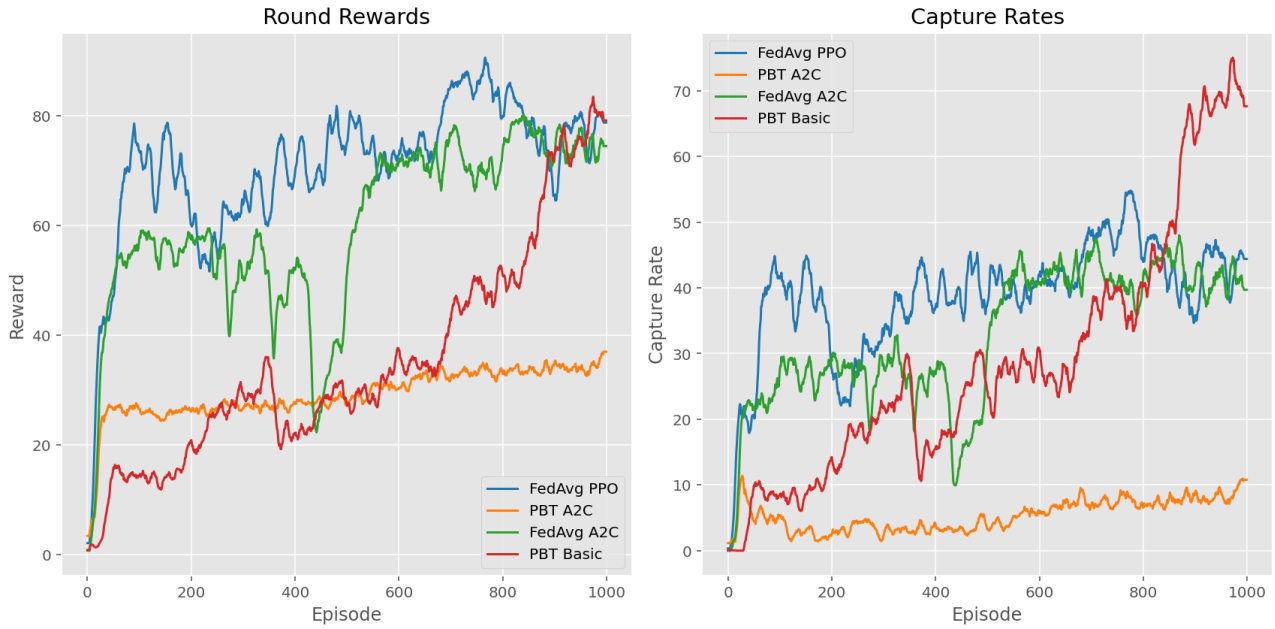


Figure 4.1: Comparison of round rewards and capture rates across training configurations.

Table 4.1: Final average reward and capture rate across training configurations after 1,000 communication rounds.

Configuration	Avg Reward	Capture Rate (%)
FedAvg A2C	78.09	46.00
FedAvg PPO	59.08	34.00
PBT A2C	34.13	7.50
PBT Basic	74.63	60.00

- **FedAvg A2C** delivered the highest average reward and exhibited strong generalization across clients. Despite wide fluctuations during training, it consistently converged to high-performing policies, establishing a solid baseline for federated reinforcement learning.

- **PBT A2C** offered smoother learning curves but substantially lower performance. While reduced volatility was observed, the method failed to generalize effectively, resulting in low capture rates and underwhelming rewards.

- **FedAvg PPO** showed intermittent high rewards and moderately strong capture rates during training. At various points, it rivaled or even briefly exceeded the FedAvg A2C baseline in terms of raw performance. However, it failed to maintain these peaks consistently over time, with both metrics eventually plateauing below the levels achieved by the simpler A2C approach. This suggests that while the increased architectural complexity of MAPPO offers potential, it may also introduce instability or difficulty in convergence under federated conditions.

- **Basic PBT A2C** yielded strong results despite its lack of stabilization mechanisms such as GAE, FedProx and gradient clipping. It achieved the highest capture rate and the second-highest reward levels, outperforming both complex PBT A2C and MAPPO configurations. This suggests that, in some cases, the dynamic client selection of PBT may be sufficient to drive convergence without the need for regularization.

Overall, while FedAvg A2C remained the strongest in terms of reward-based performance, Basic PBT A2C demonstrated the highest task success rate. These findings highlight the potential of minimalist, selection-driven strategies and point to the importance of revisiting conventional stabilization techniques when designing federated learning strategies.

With this conclusion, we conducted an extended training session of the best-performing configuration—**PBT A2C**—to evaluate its long-term learning potential and final performance ceiling. The training was allowed to run for **10,000 communication rounds**, during which

the system exhibited stable learning dynamics and continued to improve beyond the original experiment timeline.

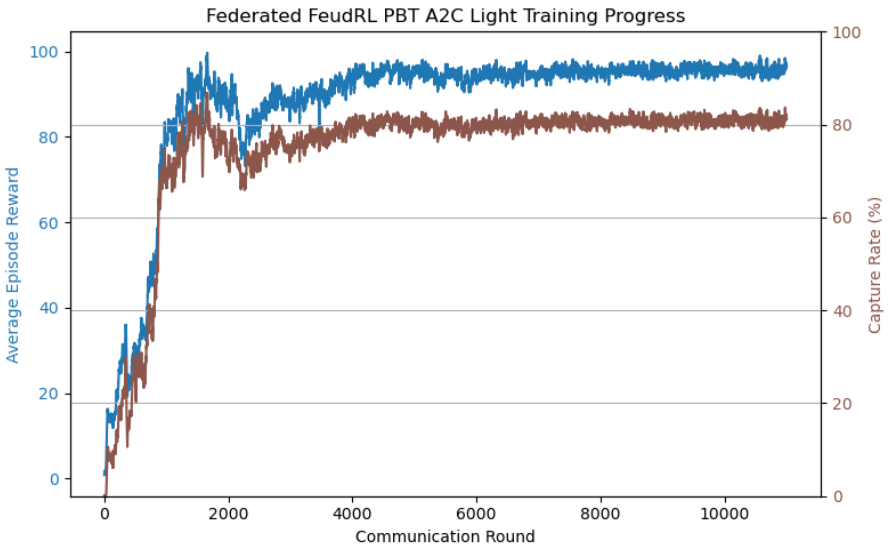


Figure 4.2: Round rewards and capture rates in an extended training for Basic PBT.

The results (see Fig. 4.2) demonstrate clear convergence behavior. By the final round, the model achieved an **average episode reward of 95.60** and a **capture rate of 80.50%**, substantially improving over the earlier 60% capture rate plateau.

2 Training Configurations Carbon Emissions

Figure 4.3 presents the environmental and computational cost metrics for each training configuration, including total CO_2 emissions, energy consumed (kWh), and runtime duration (hours). These values were obtained using the CodeCarbon framework [5].

Among all configurations, **FedAvg PPO** incurred the highest environmental and energy cost, emitting **0.41 kg CO_2**, consuming **11.71 kWh**, and requiring a total runtime of **10.76 hours**. Despite its architectural complexity, the increased energy demand was not matched by a proportional performance improvement.

In contrast, **Basic PBT A2C** was the most resource-efficient, with only 0.32 kg of CO_2 emissions, 9.06 kWh of energy usage, and a runtime of 8.33 hours. Interestingly, it achieved this while also securing the highest capture rate, indicating that simpler methods with fewer stabilizing mechanisms can lead to greener training when combined with adaptive selection strategies.

FedAvg A2C showed moderate environmental impact, with **0.33 kg CO_2** emissions, **9.57 kWh** of energy consumption, and a runtime of **8.66 hours**. While it no longer represents the

worst-case in sustainability, it remains less efficient than Basic PBT A2C.

Finally, **PBT A2C** also maintained a relatively low carbon footprint (0.32 kg CO_2, 9.24 kWh, 8.58 hours), but failed to deliver competitive performance, making it less favorable in terms of cost-effectiveness.

Overall, these results emphasize the importance of evaluating both performance and environmental sustainability in federated reinforcement learning. Basic PBT A2C stands out as the most efficient approach in this trade-off space.

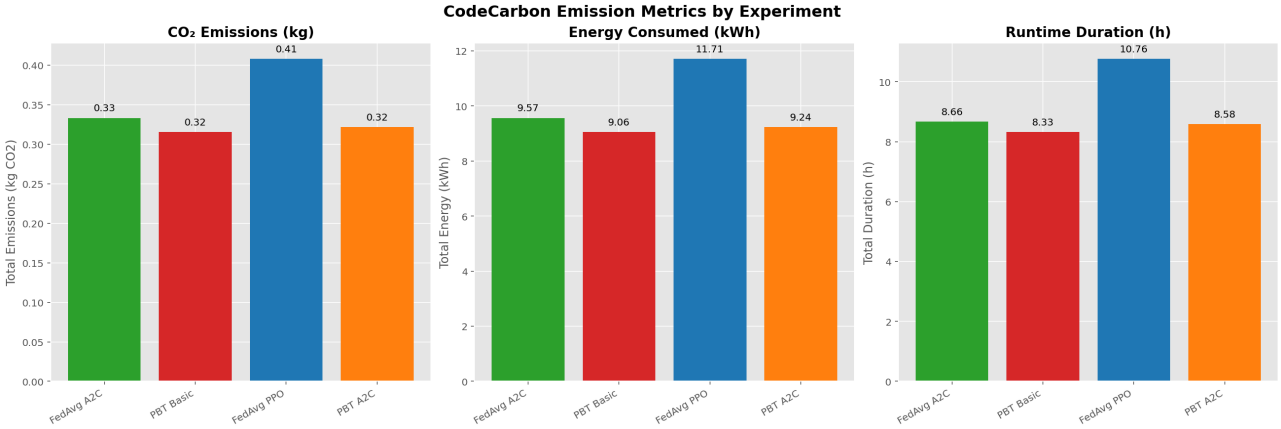


Figure 4.3: Comparison of emissions training configurations.

To better understand the environmental cost of training each configuration, we compared the carbon emissions and energy consumption to real-world activities.

The **FedAvg PPO** configuration—the most resource-intensive—consumed **11.71 kWh** of electricity and emitted approximately **0.41 kg CO_2**. For perspective:

- This is roughly equivalent to driving a car for **4 km**, based on an estimated emission of 106 g CO_2 per km [33].
- It also corresponds to **over 1.5 days** of electricity consumption in an average European household, which typically uses around **7 kWh per day** [34].

In contrast, the most efficient configuration, **Basic PBT A2C**, consumed only **9.06 kWh** and emitted **0.32 kg CO_2**. The difference of **0.09 kg CO_2** and **2.65 kWh** represents:

- Avoiding approximately **1 km** of car travel.
- Saving approximately **0.4 days** of electricity use in an average European household.

This analysis underscores that algorithmic simplification can drive both performance and sustainability. The Basic PBT configuration achieved the highest capture rate while consuming significantly less energy and producing lower carbon emissions, illustrating the value of streamlined approaches in sustainable AI development

3 Learned Behaviors and Agent Strategies

To understand how agents achieved target capture, we analyzed representative trajectory plots from evaluation episodes (Figures 4.4–4.5), illustrating both coordinated and uncoordinated behavior. These episodes reveal emergent patterns of cooperation in the manager–worker hierarchy under the basic PBT-A2C policy, which yielded the best training results. Behavior was assessed through visual inspection of model renderings over 100 randomly initialized environments.

```
Test Success Rate: 88.00% over 100 episodes.
```

This result demonstrates meaningful policy learning, though it remains below the ideal threshold of 95% success, indicating room for further optimization.

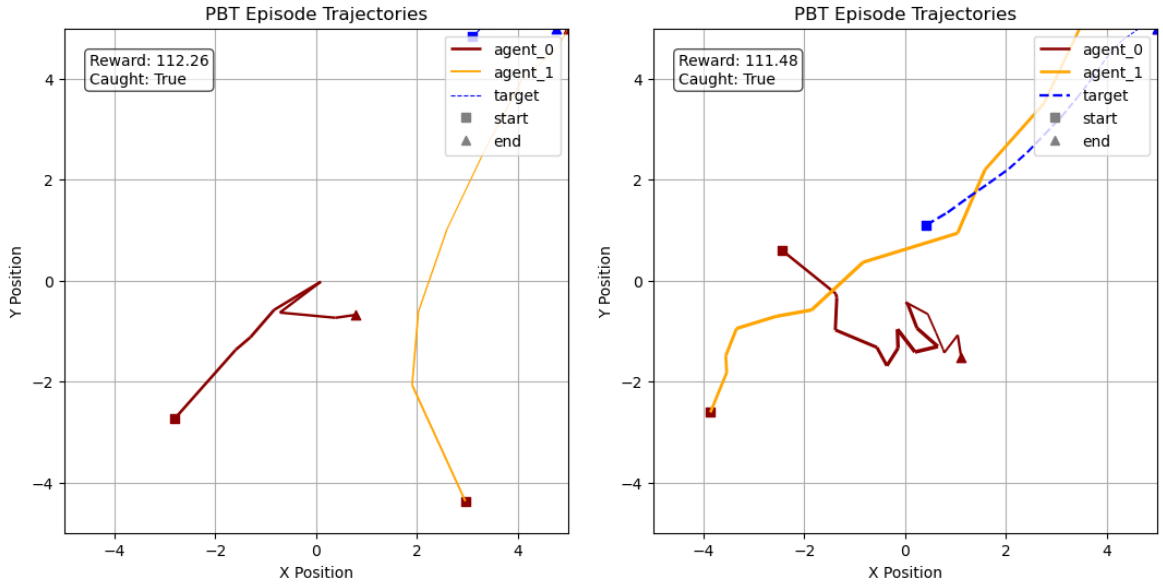


Figure 4.4: Successful episodes where agents cornered the target. One worker engaged, while the other maintained spacing.

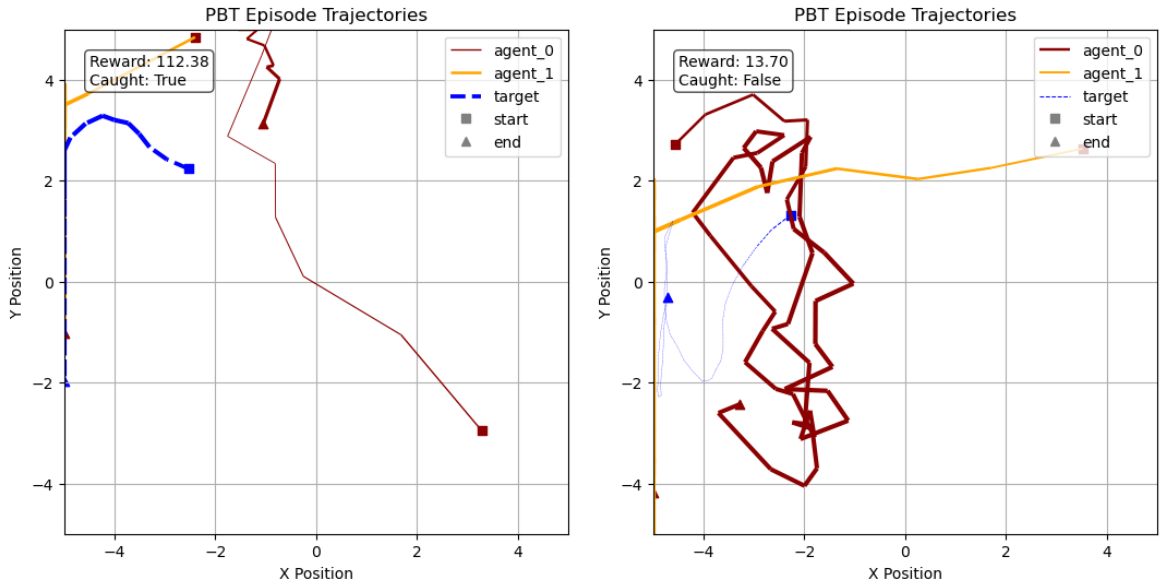


Figure 4.5: Left: Successful capture without cornering. Right: failed episode, with agents unable to constrain the target path.

The following behavioral patterns were observed:

- **Coordinated Interception:** Most captures occurred when agents executed a cornering strategy—one worker approached directly while the second blocked escape routes or remained back to prevent mutual collisions.
- **Distributed Roles:** The manager agent appeared to assign complementary roles to workers. This asymmetric behavior—where one worker leads and the other supports—helped avoid both redundancy and crashes.
- **Wall-aware Targeting:** Agents were especially successful when pushing the target toward the environment's walls. Despite the environment's design discouraging such scenarios (via repulsion mechanics), agents learned to exploit these moments for interception.
- **Cautious Z-axis Behavior:** In failed episodes, the target often escaped at very low altitudes (indicated by thinner traces). Workers appeared reluctant to pursue aggressively near the ground, suggesting learned avoidance of potential crashes—possibly influenced by the strong penalty for vertical collisions.

These strategies highlight the utility of a hierarchical manager–worker setup: while workers focus on local movement execution, the manager learns to coordinate their roles to constrain and eventually capture the evasive target. However, sensitivity to altitude and imperfect blocking sometimes led to missed captures, pointing to areas for future policy refinement.

Chapter 5

Conclusions and future work

In this final chapter, we summarize the key findings of our Federated Feudal Reinforcement Learning (FFRL) study, critically assess the extent to which the original goals were met, reflect on methodological and sustainability considerations, and outline potential for future research.

1 Conclusions of the Work

The main achievements of this project can be summarized as follows:

- **Proof of Concept for FFRL:** We have successfully designed, implemented, and evaluated a hierarchical reinforcement learning architecture that integrates Feudal RL with Federated Learning (FedAvg with FedProx and GAE). Our experiments in the multi-agent Catch environment demonstrate that the combined framework can learn coordinated manager–worker policies while preserving data locality and client privacy.

- **Empirical Performance:** Among the variants tested in a 1.000 communication round scenarios, the standard FedAvg A2C baseline achieved the highest average reward, while a basic Population-Based Training (PBT) adaptation (without GAE or FedProx) yielded the highest capture rate (60%) with lower carbon emissions (0.31 kg CO_2). These results highlight that both complex and streamlined approaches can offer distinct advantages in federated hierarchical settings.

- **Environmental Assessment:** By integrating CodeCarbon, we measured the energy and emissions footprint of each training configuration, emphasizing the trade-off between algorithmic performance and sustainability. Our findings show that simpler PBT methods can substantially reduce energy consumption and carbon output while maintaining or improving task success.

- **Emergent Strategies:** Trajectory analyses revealed that the manager effectively assigns complementary roles (catcher and blocker) to worker agents, enabling cornering strategies, eventual interceptions, and cautious altitude management.

2 Future Work

Several challenges remain ahead to validate the results presented in this work and to extend its practical applicability beyond the current scope:

1. **Real-World Robotics Deployment:** Transitioning from simulation to physical multi-robot systems introduces challenges such as sensor noise, communication delays, and hardware variability. Real-world testing is essential to assess safety, robustness, and practical feasibility.

2. **Stronger Privacy Guarantees:** While the federated setup limits data exposure, it does not offer full end-to-end privacy. Future work could explore cryptographic or differential privacy techniques to enhance client-to-manager confidentiality and regulatory compliance.

3. **Adaptive Federated Scheduling:** Introducing dynamic update schedules based on client performance, bandwidth, or energy constraints could improve scalability and responsiveness in heterogeneous or resource-constrained environments.

4. **Performance Optimization:** The current framework achieves an 80% capture rate, below the ideal 95% threshold. Further gains may come from tuning, structural changes, or hybrid learning methods.

5. **Generality Across Domains:** To validate the broader applicability of FFRL, future work should evaluate it across diverse multi-agent tasks with different coordination and reward structures.

In conclusion, this project demonstrates the feasibility and benefits of integrating hierarchical and federated paradigms in reinforcement learning. While significant progress has been made, the outlined future directions will further enhance performance, privacy, and applicability in real-world multi-agent systems.

End Matter

Bibliography

[1] Vezhnevets AS, Osindero S, Schaul T, Heess N, Jaderberg M, Silver D, et al. Feudal networks for hierarchical reinforcement learning. Preprint at arXiv. 2017 Mar 3. Available from: https://arxiv.org/abs/1703.01161 [Accessed 2025 Mar 9]

[2] McMahan B, Moore E, Ramage D, Hampson S, Arcas BA. Communication-efficient learning of deep networks from decentralized data. Preprint at arXiv. 2016 Feb 18. Available from: https://arxiv.org/abs/1602.05629 [Accessed 2025 Mar 9]

[3] Li T, Sahu AK, Talwalkar A, Smith V. Federated optimization in heterogeneous networks. Preprint at arXiv. 2020 Dec 13. Available from: https://arxiv.org/abs/1812.06127 [Accessed 2025 Mar 9]

[4] United Nations. Sustainable development goals [Internet]. New York: United Nations. Available from: https://sdgs.un.org/goals [Accessed 2025 Mar 9]

[5] CodeCarbon. CodeCarbon: track and reduce your AI carbon footprint [software]. Available from: https://codecarbon.io/ [Accessed 2025 Mar 9]

[6] Agile Alliance. Manifesto for agile software development [Internet]. 2001. Available from: http://agilemanifesto.org/ [Accessed 2025 Mar 9]

[7] Beck K. Extreme programming explained: embrace change. 2nd ed. Boston (MA): Addison-Wesley; 2004. ISBN: 978-0321278654.

[8] Anaconda, Inc. Anaconda: Python for data science [software]. Austin (TX): Anaconda, Inc. Available from: https://www.anaconda.com/ [Accessed 2025 Mar 9]

[9] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst. 2019;32. Available from: https://pytorch.org/ [Accessed 2025 Mar 9]

[10] Pygame. Pygame: Python game development [software]. Available from: https://www.pygame.org/ [Accessed 2025 Mar 9]

[11] Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: a next-generation hyperparameter optimization framework. Preprint at arXiv. 2019 Jul 24. Available from: https://arxiv.org/abs/1907.10902 [Accessed 2025 Mar 9]

[12] Harris CR, Millman KJ, van der Walt SJ, et al. Array programming with NumPy. *Nature*. 2020 Sep 16;585(7825):357-62.

[13] Terry J, Black B, Hari A, et al. PettingZoo: a Python library for multi-agent reinforcement learning environments [software], 2020. Available from: https://pettingzoo.farama.org/ [Accessed 2025 Mar 9]

[14] Felt F. CrazyRL [software]. GitHub repository. Available from: https://github.com/ffelten/CrazyRL [Accessed 2025 Mar 9]

[15] Lamport L. LaTeX: a document preparation system. 2nd ed. Reading (MA): Addison-Wesley; 1994. ISBN: 978-0201529838.

[16] Schulman J, Moritz P, Levine S, Jordan M, Abbeel P. High-dimensional continuous control using generalized advantage estimation. Preprint at arXiv. 2016 Jun 7. Available from: https://arxiv.org/abs/1506.02438 [Accessed 2025 Mar 9]

[17] Bengio Y, Louradour J, Collobert R, Weston J. Curriculum learning. Proc Mach Learn Res. 2009;5:41-8. Available from: https://dl.acm.org/doi/10.1145/1553374.1553380

[18] Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks. Preprint at arXiv. 2013 Nov 21. Available from: https://arxiv.org/abs/1211.5063 [Accessed 2025 Mar 9]

[19] Bellman R. Dynamic Programming. Princeton (NJ): Princeton University Press; 1957.

[20] Sutton RS, Barto AG. Reinforcement Learning: An Introduction. Cambridge (MA): MIT Press; 2000.

[21] Watkins CJCH, Dayan P. Q-learning. Machine Learning. 1992;8:279-292.

[22] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529-533. Available from: https://www.nature.com/articles/nature14236 [Accessed 2025 Mar 9]

[23] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. Preprint at arXiv. 2017 Jul 20. Available from: https://arxiv.org/abs/1707.06347 [Accessed 2025 Mar 9]

[24] Zhuo H, Wang Z, Zhu Y, Pu S, Tan M. Federated reinforcement learning. Preprint at arXiv. 2019 Jan 24. Available from: https://arxiv.org/abs/1901.08277 [Accessed 2025 Apr 15]

[25] Jaderberg M, Dalibard V, Osindero S, Czarnecki WM, Donahue J, Razavi A, Vinyals O, Green T, Dunning I, Simonyan K, Kavukcuoglu K. Population Based Training of Neural Networks. Preprint at arXiv. 2017 Jul 17. Available from: https://arxiv.org/abs/1711.09846 [Accessed 2025 Apr 15]

[26] Chen H, Liang W, Li K, Ma Z, Zhang Y, Li K, Lin X. FedPop: Federated Population-Based Hyperparameter Tuning. Preprint at arXiv. 2023 Aug 16. Available from: https://arxiv.org/abs/2308.08634 [Accessed 2025 Apr 15]

[27] Huang C, Ma W, Wang J, Wang H. Neural networks with motivation. Preprint at ResearchGate. 2021 Jan. Available from: https://www.researchgate.net/publication/348428396_Neural_Networks_With_Motivation [Accessed 2025 Apr 15]

[28] Sutton RS, Precup D, Singh S. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. Artificial Intelligence. 1999;112(1-2):181-211. Available from: https://doi.org/10.1016/S0004-3702(99)00052-1 [Accessed 2025 Mar 9]

[29] Kulkarni TD, Narasimhan K, Saeedi A, Tenenbaum JB. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence; 2016. Available from: https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/viewPaper/12389 [Accessed 2025 Mar 9]

[30] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. Journal of Machine Learning Research. 2012;13:281-305. Available from: http://www.jmlr.org/papers/volume13/bergstra12a.html [Accessed 2025 Mar 9]

[31] Snoek J, Larochelle H, Adams RP. Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems; 2012. Available from: https://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf [Accessed 2025 Mar 9]

[32] Debés M. Federated FeudRL [software]. GitHub repository. Available from: https://github.com/mdebes/Federated_FeudRL [Accessed 2025 May 31]

[33] European Environment Agency. CO_2 emissions performance of new passenger cars in Europe. 2024 Dec 16. Available from: https://www.eea.europa.eu/en/analysis/indicators/co2-performance-of-new-passenger [Accessed 2025 May 16]

[34] Eurostat. Electricity consumption in households per capita. 2023. Available from: https://ec.europa.eu/eurostat/databrowser/view/ten00117/ [Accessed 2025 May 16]

Annex

1 The Bellman Equation and the Optimal Value Function

Bellman Equation

The Bellman Equation is a core concept in Reinforcement Learning which provides a **recursive** definition of the value of a state under a given policy. For a policy π, the state-value function is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\Big[R(s,a) + \gamma V^{\pi}(s')\Big], \tag{5.1}$$

In plain language, this equation states that the value $V^{\pi}(s)$ of a state s (i.e., the long-term expected reward when starting in s) is equal to the immediate reward $R(s,a)$ received after taking an action a, plus the discounted value of the subsequent state s'. Here, the discount factor $\gamma \in [0,1)$ determines how much future rewards are worth relative to immediate rewards. The expectation $\mathbb{E}_{\pi}$ indicates that we average over all possible actions and the resulting state transitions according to the policy π and the environment's dynamics (see [19, 20]). Essentially, this formula encapsulates the idea that today's value is built on both the reward obtained immediately and the value of tomorrow's state.

Figure 5.1 illustrates the recursive structure of the Bellman equation for the state-value function $V^{\pi}(s)$. The agent selects an action according to policy $\pi(a|s)$, receives a reward $R(s,a)$, transitions to a new state s' via $P(s'|s,a)$, and accumulates future expected value $V^{\pi}(s')$. This diagram emphasizes the value update based on immediate and future rewards.

Optimal Value Function

The Optimal Value Function $V^{\pi^*}(s)$ takes this idea one step further. It represents the best possible value that can be obtained starting from state s, assuming that the agent follows the

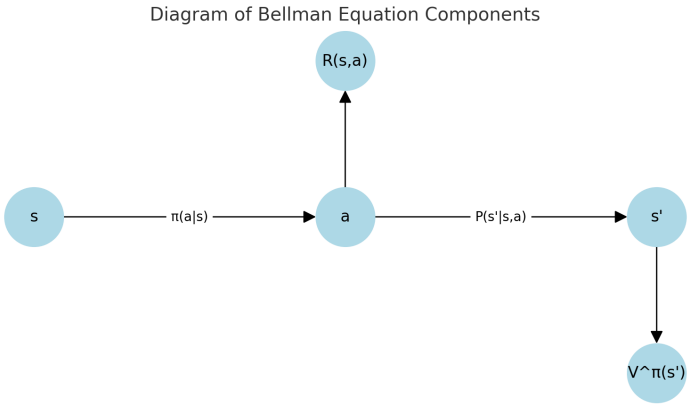


Figure 5.1: Recursive structure of the Bellman equation under policy π. Author's own work.

optimal policy π^*. The optimal value function is given by:

$$V^{\pi^*}(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi^*}(s') \right\}. \tag{5.2}$$

In simple terms, this equation tells us that the optimal value $V^{\pi^*}(s)$ is achieved by choosing the action a that maximizes the sum of the immediate reward $R(s,a)$ and the discounted expected future rewards. The term $\sum_{s'} P(s'|s,a)V^{\pi^*}(s')$ represents the average value over all possible next states s', weighted by the probability $P(s'|s,a)$ of reaching each state when action a is taken. This formulation is fundamental in reinforcement learning as it defines what it means to follow an optimal policy—by always selecting the action that leads to the highest possible long-term benefit.

Both equations lay the foundation for various RL algorithms. The Bellman Equation (Equation 5.1) forms the basis for iterative methods like value iteration and policy evaluation, while the Optimal Bellman Equation (Equation 5.2) is at the heart of algorithms such as Q-learning, which seeks to learn $V^{\pi^*}(s)$ or its action-value equivalent $Q^{\pi^*}(s,a)$ through repeated updates.

These equations not only provide the mathematical framework for predicting future rewards but also encapsulate the core idea of recursive decision making in environments where outcomes are uncertain. Their development has been instrumental in the evolution of both classical and deep reinforcement learning methods.

Hyperparameters table

Hyperparameter	FedAvg A2C	FedAvg PPO	PBT A2C	Basic PBT	Search Range
Manager hidden size	64	64	128	192	{64, 128, 192, 256}
Worker hidden size	192	192	256	256	{64, 128, 192, 256}
Manager learning rate	3.6×10^{-5}	0.0003	6.5×10^{-5}	2.6×10^{-5}	$[1 \times 10^{-5}, 5 \times 10^{-4}]$ (log)
Worker learning rate	0.00015	1.16×10^{-5}	1.95×10^{-5}	1.92×10^{-5}	$[1 \times 10^{-5}, 5 \times 10^{-4}]$ (log)
Entropy coefficient	0.013	0.018	0.012	0.02	$[0.001, 0.1]$
Discount factor γ	0.97	0.94	0.905	0.915	$[0.90, 0.99]$
FedProx coefficient μ	0.0003	9.37×10^{-5}	0.0002		$[1 \times 10^{-5}, 5 \times 10^{-4}]$ (log)
GAE lambda λ	0.92	0.92	0.92		$[0.90, 0.99]$
Communication rounds	1000	1000	1000	1000	
Episodes per round	50	50	50	50	

Table 5.1: Final hyperparameter values and their corresponding search ranges used in Optuna tuning.

2 Logging, Model Saving, and Evaluation Utilities

To support analysis, reproducibility, and post-training validation, the implementation includes a set of practical utilities for experiment logging, model persistence, and evaluation. This logging infrastructure forms the foundation of our methodology, as it enables us to track performance across trials, test competing hypotheses, and systematically refine architectural and training decisions. Nearly all insights described in this chapter—from hyperparameter tuning to stability diagnostics—are grounded in the structured data captured during these logging routines.

- **Hyperparameter Logging:** After training, the system appends the full set of used hyperparameters and the average episode reward over the final communication rounds to a structured CSV-like text file. This facilitates performance tracking and comparison across different runs and configurations. Example entries are shown in Table 5.2

- **Model Serialization:** The final global manager and worker networks are saved to disk using PyTorch's native serialization format. This includes both policy and value networks, enabling full checkpoint recovery and reuse for testing or further training.

- **Real-Time Evaluation:** A visual testing utility renders agent behavior in a held-out environment using Pygame. The manager and worker agents operate in inference mode over 20 randomly initialized episodes. Each episode is monitored for successful target interception, determined via a custom `info["caught"]` flag returned by the environment.

m_hid	w_hid	m_lr	w_lr	H	γ	R	E	μ	λ	Avg. Reward
256	128	1e-5	2.5e-5	0.0044	0.90	500	50	2.3e-5	0.95	90.25
256	128	1e-5	2.5e-5	0.0044	0.90	1000	50	2.3e-5	0.95	73.29
256	128	1e-5	3.8e-5	0.0190	0.90	500	50	1.6e-5	0.90	60.75
256	192	1e-5	1e-4	0.0010	0.93	500	50	3.7e-5	0.95	12.20
128	192	7.96e-5	1.31e-5	0.0400	0.9735	1000	50	1.26e-5	0.91	53.28
64	192	7.96e-5	1.31e-5	0.0400	0.9735	1000	50	1.26e-5	0.91	52.48

Table 5.2: Selected experiment log entries showing hyperparameter settings and resulting average reward over the final 100 rounds. Abbreviations: m_hid = manager hidden size, w_hid = worker hidden size, m_lr = manager learning rate, w_lr = worker learning rate, H = entropy coefficient, R = communication rounds, E = episodes per round, μ = FedProx coefficient, λ = GAE parameter.

- **Training Metrics Visualization:** After training, the system produces plots showing the progression of average episode rewards, capture rates, and average steps per episode over communication rounds. These plots offer intuitive insights into convergence behavior and policy quality.

Together, these tools provide a reliable infrastructure for experimentation and facilitate both qualitative and quantitative validation of the learned federated feudal policies.

3 Algorithms

Algorithm 1 Learning Rate Scheduling per Communication Round

Initialize $lr_{manager} \leftarrow 0.001$
Initialize $lr_{worker} \leftarrow 0.001$
Set decay interval $d \leftarrow 100$
Set decay factor $\gamma \leftarrow 0.5$
for $r = 1$ to R **do**
  Run E local episodes per client
  Perform federated aggregation
  if $r \bmod d = 0$ **then**
    $lr_{manager} \leftarrow \gamma \cdot lr_{manager}$
    $lr_{worker} \leftarrow \gamma \cdot lr_{worker}$
  end if
end for

4 GAE and FedProx Synergy:

The combination of Generalized Advantage Estimation (GAE) and FedProx was found to significantly enhance training stability and robustness. GAE reduced the variance in policy gradient estimates, leading to smoother and more stable updates within local trajectories. Simultaneously, FedProx introduced a regularization term, scaled by a coefficient $\mu > 0$, that constrained the drift of local worker models from the global baseline — an essential property in federated settings with client heterogeneity or intermittent performance degradation. The regularization strength μ controls the trade-off between local optimization flexibility and global consistency. Empirical results indicated that using both techniques together improved sample efficiency and led to more reliable convergence, particularly when clients operated under different environmental dynamics or hardware limitations.

Algorithm 2 Local Update with GAE and FedProx Regularization

Initialize local worker parameters θ_i
Receive global parameters θ_{global}
for each local episode **do**
  Collect trajectory $\tau_i = \{(s_t, a_t, r_t)\}_{t=1}^{T}$
  Estimate advantages $\hat{A}_t$ using GAE with λ
  Compute policy gradient loss $L^{\text{PG}}(\theta_i)$
  Compute value function loss $L^{\text{VF}}(\theta_i)$
  Compute FedProx regularization: $L^{\text{prox}}(\theta_i) = \mu\|\theta_i - \theta_{global}\|^2$
  Total loss: $L(\theta_i) = L^{\text{PG}} + L^{\text{VF}} + L^{\text{prox}}$
  Update $\theta_i \leftarrow \theta_i - \alpha\nabla_{\theta_i} L(\theta_i)$
end for

Algorithm Walkthrough: Given the complexity introduced by the integration of GAE and FedProx in the local training procedure, the following breakdown clarifies each step. This walkthrough outlines the local training phase executed independently on each client. Each variable and computation used in the pseudocode is explained below for clarity:

- θ_i: the current set of parameters for the local worker model on client i.
- θ_{global}: the latest global model parameters received from the server.
- $\tau_i = \{(s_t, a_t, r_t)\}_{t=1}^{T}$: the trajectory (sequence of states, actions, and rewards) collected during a single episode of interaction with the environment.
- $\hat{A}_t$: the estimated advantage at time step t, computed using Generalized Advantage Estimation (GAE), which leverages both immediate and future rewards to assess the value of each action taken.

- $L^{\text{PG}}(\theta_i)$: the policy gradient loss, which encourages the model to reinforce actions that lead to higher estimated advantages.
- $L^{\text{VF}}(\theta_i)$: the value function loss, used to train the model to predict expected returns accurately.
- $L^{\text{prox}}(\theta_i) = \mu\|\theta_i - \theta_{global}\|^2$: the FedProx regularization term, which penalizes local updates that deviate too far from the global model. The coefficient μ controls the strength of this penalty.
- $L(\theta_i) = L^{\text{PG}} + L^{\text{VF}} + L^{\text{prox}}$: the total loss function, combining learning objectives and regularization into a single optimization target.
- α: the local learning rate, which determines the step size used in the update.
- $\nabla_{\theta_i} L(\theta_i)$: the gradient of the total loss with respect to the local parameters.
- $\theta_i \leftarrow \theta_i - \alpha \nabla_{\theta_i} L(\theta_i)$: the local update step that adjusts the parameters to reduce the total loss.

This process is repeated for each local episode, allowing the client to refine its local model before contributing to the global aggregation. GAE improves learning signal quality, while FedProx ensures local updates remain compatible with global coordination.

Computational Considerations: Among the various components of the local training procedure, the combination of Generalized Advantage Estimation (GAE) and FedProx regularization constitutes the most computationally intensive stage. GAE involves per-step backward computations over each trajectory to estimate advantages, while FedProx adds a per-parameter regularization term that increases the gradient update complexity. Since both operations are performed repeatedly across all local episodes on each client, their cumulative cost exceeds that of one-time operations like global model averaging.