The Pennsylvania State University

The Graduate School

VALUE ALIGNMENT IN ETHICAL DILEMMAS THROUGH REINFORCEMENT LEARNING

A Thesis in

Aerospace Engineering

by

Arthur Melo Cruz

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

December 2023

The thesis of Arthur Melo Cruz was reviewed and approved by the following:

Alan R. Wagner

Associate Professor of Aerospace Engineering

Thesis Advisor

Jacob W. Langelaan

Professor of Aerospace Engineering

Amy R. Pritchett

Professor of Aerospace Engineering

Department Head

Abstract

The principle of value alignment [1] suggests that autonomous decision-making should be aligned with the values of humans impacted by the system's decisions. Following this principle, methods that allow an agent to map moral value systems to decision-making in ethical dilemmas are proposed. The methods are tested on two ethical dilemma scenarios that have no universally agreed upon solutions: a home invasion scenario, and a search and rescue scenario. For the home invasion scenario, the proposed architecture leverages Inverse Reinforcement Learning (IRL) to allow an agent to learn different ethical behaviors from human demonstrations. The system is used to train an agent to generate different policies when facing the threat of a home break-in by a hostile actor. Testing of the trained policies show convergence over three different human-demonstrated policies (hide, call the police, and kill the invader) for decision-making, with the agent first deciding what to do based on contextual input, and subsequently interacting with the environment to execute its decision. For the search and rescue scenario, the proposed architecture aimed to achieve value alignment by codifying a moral value system directly in the reward function through handcrafted reward features and weights, without observation of a human ethical exemplar. The system is used to train an agent to rescue a number of hostages from a threat actor, while following the moral directive of not killing the threat actor. The trained policy demonstrated a strong trade-off between performance in the search and rescue task and value alignment with the moral directive of not killing the threat actor, with the agent only being able to achieve value alignment while reducing the number of rescued hostages. The policy was also tested against variations in environment configurations in order to evaluate generalizability of the solution, with the results indicating that small variations in scenario configurations lead to significant decrease in performance. The results from both experiments demonstrate that the proposed architectures are capable of producing value-aligned ethical behavior under specific conditions and can, perhaps, serve as a stepping stone for future architectures that address decision-making in ethical dilemma scenarios.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. Alan Wagner, for his thoughtful guidance, patience, and expertise throughout this rewarding journey. Dr. Wagner's mentorship has been invaluable in shaping the direction and quality of this thesis. I am also profoundly thankful to my parents, Josemar and Tereza Cassia, and my brother, Daniel, whose boundless love and encouragement have been my constant motivation. To my significant other, Yoon Ji Huh, your unwavering support, understanding, and belief in me have been a source of strength, and I am incredibly lucky and blessed to have you by my side. I want to extend my appreciation to my dear friends, especially Karla, Mostafa and Gala, who have provided a supportive network of encouragement, camaraderie, and emotional support. Your presence in my life has made this endeavor all the more enjoyable. This thesis would not have been possible without the collective support of these incredible individuals, and for that, I am eternally grateful.

This material is based upon work supported by the National Science Foundation under Grant No. 1849068 and 1848974. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Chapter 1

Introduction

1.1 Motivation

The field of Machine Ethics seeks to develop artificially intelligent systems that can behave ethically in real-world scenarios [2]. When agents and robots must make ethically difficult decisions, it is critical that these systems act according to ethical norms and values. One concept for creating ethical behavior in an intelligent system is value alignment [1, 3], which directs that AI decision-making should be aligned with the values of the humans involved within a given scenario. Yet, because values may differ substantially across cultures, demographics, and time, it is very difficult to create a one-size-fits-all system that produces value-aligned decisions when faced with an ethical dilemma.

Ethical or moral dilemmas are situations in which persons, robots or virtual agents are required to make one of at least two available moral decisions, and cannot make multiple decisions at the same time [4]. By making a single decision, the agent honors a set of moral values that are associated with that decision, while also violating moral values that are associated with the other available decisions that could have been made.

The classic Trolley Problem [5], for example, describes an ethical dilemma in which a trolley operator is responsible for steering a malfunctioning trolley which cannot be

stopped. The unstoppable trolley must be steered in the direction of one of two paths: in one path, the trolley runs over and kills a set of persons; in the other path, the trolley runs over and kills a different set of persons. This scenario is an ethical dilemma because the trolley operator is morally required to choose one path for the trolley, thus unavoidably killing at least one set of persons.

By definition, ethical dilemmas do not have globally agreed-upon best solutions [6]. In this context, best solutions are solutions that are perfectly aligned with all moral values or ethical frameworks involved in the scenario, without violations. Each action or decision that can be made in the dilemma may serve as a solution for the dilemma according to a particular set of moral values, thus being aligned with those values, but no action can be deemed as globally value-aligned. This condition makes it difficult for artificially intelligent agents or robots to make decisions in a dilemma that are based on global or absolute moral values, but it does not prohibit the agent from following locally value-aligned solutions that follow specific sets of moral values or ethical guidelines.

It is possible to illustrate the applicability of local value alignment through a particular version of the Trolley Problem. In this version, the trolley operator must decide between killing a child as the first choice or killing an 80-year-old adult as the second choice. Suppose the trolley operator believes that the child's life should be prioritized, since the child has a potentially longer remaining lifespan than the 80-year-old. In that regard, the value-aligned choice would be to kill the 80-year-old instead of the child. Now, suppose that the operator values the wisdom, experience and memories accumulated by the 80-year-old. In that case, the value-aligned choice would be to kill the child. In this example, the two solutions are both locally value-aligned with their respective moral value systems, yet they are conflicting with each other and cannot fully honor all moral values involved and be classified as globally aligned.

Ethical dilemmas are ubiquitous in aerospace and defense contexts. An example of

a famous dilemma was the bombing of Dresden in World War II, in which the United States and Britain intentionally bombed the city of Dresden to completely destroy it [7]. In that episode, U.S. and British military leadership valued their security and victory more than preservation of Dresden's population and art treasures, with the bombings destroying strategic targets at the cost of over 20,000 civilian lives and loss of cultural heritage. The decision-making agents in this scenario were humans, but it is not difficult to imagine a future in which a fully autonomous UAV is making such life-or-death decisions, without input from a human operator. In fact, the U.S. Air Force's XQ-58A Valkyrie program [8] is already demonstrating prototypical architectures for high-level autonomous decision-making in the battlefield. As adoption of autonomy in all levels of decision-making continues to increase, it is important to consider how autonomous agents should make decisions in ethical dilemmas.

1.2 Objectives of This Work

Research Question: How can one design a system that uses reinforcement learning to produce value-aligned decisions in an ethical dilemma?

This work seeks to evaluate whether it is possible to design a machine learning system based on reinforcement learning that makes value-aligned decisions in an ethical dilemma. Based on a novel formalism presented on Chapter 3, the system would model a function that maps moral value systems to decisions in a moral dilemma. This system would be considered successful in achieving value alignment if the decisions produced are equal to the decisions made by an independent human counterpart.

This thesis is scoped around scenarios in which the agent's states are fully observable, meaning that the agent has access to all state variables at any given moment. This condition eliminates uncertainty from the agent's decision-making process, which permits

prediction of future states based on decisions made in the current or past states. In the context of ethical dilemmas, this scope excludes dilemmas in which the agent does not know who will be affected by its decisions, as well as how they will be affected. For example, a modified trolley problem in which the agent does not know how many people would be killed in each of the tracks is not included in this scope.

1.3 Experiments and Proposed Architectures

This work explores reinforcement learning architectures for decision-making applied to two dilemmas: one in which an agent has to manage a home invasion, and another in which an agent must execute a search and rescue operation. In the home invasion scenario, the agent represents a domestic robot that is tasked with deciding what to do when an invader breaks into its home. The agent must choose one of four possible actions: remain idle and do nothing, kill the invader, call the police, or hide from the invader. In the search and rescue scenario, the agent represents a rescue robot that must retrieve a number of hostages in the environment, while having to deal with a threat that may compromise the rescue operation. The agent must choose between picking up a weapon and killing the threat, or simply dodging the threat while rescuing the hostages.

Two different learning architectures are proposed and evaluated using the scenarios. For the home invasion scenario, a dual-layer architecture based on inverse reinforcement learning is proposed. For the search and rescue scenario, a reinforcement learning architecture based on proximal policy optimization is proposed. Each architecture presents a unique approach to achieve value alignment in the scenarios. For the home invasion scenario, the proposed architecture infers value alignment by observing an ethically competent human expert. In the search and rescue scenario, the proposed architecture relies on reward function tuning for achieving value alignment.

In addition to evaluation of value alignment, generalizability of the resulting policy

particularly in the search and rescue scenario is also explored. Policy generalizability testing is an important for measuring of robustness of learned behaviors, since it indicates how well the policy performs in conditions that are different from those used for training. After initial evaluation of policy performance against standard trained conditions, the architecture used in the search and rescue scenario is tested against scenario variations in order to map out which of those variations influence policy performance.

Reinforcement learning (RL) was chosen as a conceptual foundation for this work due to its reward-based structure. I believe that the reward function in the Markov Decision Process (MDP) can be adapted to the ethical dilemma formalism. The two proposed architectures aim to allow the agent to produce value-aligned decisions that can be used allowing the agent to plan a path to achieve the decision. The home invasion and search and rescue scenarios were built using a low-fidelity grid map that is representative of the dilemmas in question, and the agent is responsible for devising trajectories across the map that result in value-aligned solutions to the dilemmas.

1.4 Reader's Guide

The remainder of this thesis is composed of five chapters. Chapter 2 provides an overview of prior efforts in machine ethics and AI communities on developing value-aligned intelligent agents. Chapter 3 covers the methods that formalize the decision-making process in the ethical dilemma, serving as the foundation for reinforcement learning and inverse reinforcement learning techniques. Chapters 4 and 5 present the proposed architectures and experimental applications of the two dilemma scenarios covered: home invasion (Chapter 4) and search and rescue (Chapter 5). Lastly, Chapter 6 discusses the lessons learned from the two experiments and next steps for further development.

Chapter 2

Current State-Of-The-Art

This chapter presents prior research related to machine ethics and artificial intelligence exploring the architectures, mechanisms and techniques that enable artificial agents to make value-aligned decisions. An overview of rules-based and learning-based methods is provided, along with specific examples of successful implementations of proposed architectures. Lastly, the current gaps in the literature are identified, with discussion of how this work aims to fill such gaps.

2.1 Value Alignment in Artificial Intelligence

The concept of value alignment as a guiding principle for developing ethical intelligent agents has been widely explored in the machine ethics and AI communities [1, 3, 9]. Different technical approaches have been proposed to ensure that an intelligent agent's behavior conforms to social norms and ethical guidelines. One approach is rules-based [10–14], in which rules and directives are codified into the intelligent agent in advance. Another competing approach focuses on learning-based techniques [1, 15–17], in which the intelligent agent learns an objective function to act ethically while interacting with the environment or training from curated datasets.

2.1.1 Rules-Based Value Alignment

Rules-based value alignment has been proposed as a way to constrain the actions of autonomous agents so that they do not violate laws, ethical boundaries or norms of engagement [18–20]. With this approach, specific constraints are programmed into the agent's policies and decision-making architecture, precluding any actions that would result in value misalignment. The strength of this approach centers around the establishment of legal and moral bases for the autonomous agent's behavior, which increases explainability of the agent's actions and reduces the risks of rogue behavior [21]. On the other hand, a disadvantage of this approach relates to scalability, adaptability, and generalizability, as the number of necessary hard-coded rules may be high and the rules might change over time and across cultures and geographies.

A series of rules-based architectures have been proposed to guide the development of value-aligned autonomous agents in diverse contexts. Arkin [13] has proposed an extension of existing autonomy architectures for ensuring that autonomous lethal action abides by Laws of War and Rules of Engagement, specifically in battlefield scenarios. Bringsjord et al. [14] suggested a deontic logic-based approach for governing autonomous behavior in generalized scenarios. Briggs and Scheutz [10] argued for mechanisms to determine when and how a robot should reject directives from a human in generalized scenarios.

2.1.2 Learning-Based Value Alignment

Advances of machine learning techniques in recent years have led to a surge of learning-based methods for value alignment [22]. These methods build upon big data or statistical mechanisms to train an agent on how to act ethically and achieve value alignment. Different machine learning techniques have been leveraged within learning-based architectures,

and two of the most popular are reinforcement learning (RL) and inverse reinforcement learning (IRL) [23–26]. In the context of RL and IRL, the agent learns a specific behavior (known as a policy) by being rewarded or punished across multiple cycles of trial and error in an environment [27]. In RL, the reward function that dictates how the agent is rewarded or punished is designed, and in IRL the reward function is unknown and must be discovered or estimated.

One disadvantage of architectures based on reinforcement learning is the difficulty of designing reward functions that lead the agent to learn value-aligned behavior [1, 28]. Reward function designs must be robust enough to prevent the agent from deviating from desired value-aligned behavior, as well as prevent specification gaming [29]. Inverse reinforcement learning potentially solves this problem, as the learning architecture infers the reward function from sources of information such as observation and imitation instead of relying on fine-tuned rewards that are not easily designed [30].

When it comes to techniques for executing IRL, one approach relies on observing expert behavior in order to estimate a linear reward function. This function then leads to generation of a policy which attains performance close to the expert's by minimizing the distance between expert- and agent-generated feature expectations [15]. Other methods involve maximizing the entropy of the distribution over behaviors [31], acquiring the posterior over hypotheses using Bayes Rule [32], and learning a model through regression or classification that imitates observed behavior [33, 34].

Several approaches have attempted to leverage combinations of reinforcement learning, inverse reinforcement learning and related machine learning frameworks to propose architectures that produce ethical decisions [23–26, 35, 36]. Most machine ethics architectures focus on achieving optimal ethical policies (i.e. policies that maximize cumulative rewards for a predetermined ethical behavior) in scenarios that have a single ethical courses of action. Following this approach, Abel et al. [23] propose a POMDP (Partially Observable

Markov Decision Process) framework for the generation of optimal ethical behavior. Similarly, Wu and Lin [24] propose a system that minimizes the chance of unethical behavior by incorporating human action data into a regular reward function to guide the reinforcement learning agent. Rodriguez-Soto et al. [35] also introduce a related scheme which offers theoretical guarantees for the creation of environments wherein agents learn to behave ethically.

A different approach mixes learning- and rules- based methods by adding ethical constraints to the agent's learned actions to guarantee that it behaves ethically. Loreggia et al. [36] have argued for using Ceteris Paribus Networks (CP-nets) constrained by ethical principles to learn behavior or choice preferences. Balakrishnan et al. [25] and Noothigattu et al. [26] have focused on generating constrained ethical RL policies that follow a specific ethical directive.

2.2 Gaps in Literature

Prior approaches have generally been demonstrated and evaluated on a range of ethics related toy problems. The application of IRL and pure reinforcement learning in these problems is successful in part because these methods are used to find optimal policies that follow a single objective. Yet for true ethical dilemmas that allow multiple competing decisions, there is no single objective or resulting policy that can be used to represent the many different value systems people use to evaluate a dilemma. **This thesis, in part, seeks to create an agent that can learn particular value systems and use what it has learned to make difficult ethics-related decisions.** This problem may require the agent to go beyond single-policy optimization because, for an ethical dilemma, there is no single policy to be optimized.

In the toy problems covered by prior work, not all the actions available by agents are moral actions, so they do not constitute true ethical dilemmas. For example, in

the "Cake or Death" problem described by Abel et al. [23], the action of killing three people cannot be in any sense regarded as moral when comparing to the isolated action of baking a cake. In that case, there exists only one clear moral policy, which is to always choose the action of baking the cake instead of killing. Since there is only one moral option/policy available, the scenario is not a true moral dilemma. The "Burning Room" scenario covered by the same authors also leads to a single best policy, despite offering multiple courses of action, so it also does not constitute a true ethical dilemma. All prior work cited include scenarios that offer clear-cut single best policies for ethical behavior, thus none of them address true ethical dilemmas.

This thesis aims to fill the gap present in the literature by addressing scenarios in which multiple conflicting ethical decisions can be made in true moral dilemmas. In the scenarios covered here, different value systems may lead artificial agents to make different decisions depending on the dominant value system at play. The architectures proposed by this work aim to demonstrate that agents can learn specific behaviors that are value-aligned with specific value systems, thus locally solving ethical dilemmas without attempting to find global solutions that may be impossible to achieve.

Chapter 3

Methods

This chapter presents previously developed methods necessary for training artificial agents using reinforcement learning and inverse reinforcement learning, as well as a newly developed formalism for modeling the decision-making process in an ethical dilemma. The Markov Decision Process (MDP) is described, establishing the basis upon which all reinforcement learning techniques are built. Subsequently, two methods for solving MDPs are explored: approximate Q-learning and proximal policy optimization. On the Inverse Reinforcement Learning (IRL) front, the underpinnings for estimating reward functions from feature expectations are presented. Lastly, an ethical dilemma formalism is introduced, laying the groundwork for how agents can map moral value systems to decisions in an ethical dilemma.

3.1 Markov Decision Process

We use Markov Decision Processes (MDP) to guide the agent's behavior when faced with an ethical dilemma. An MDP is commonly represented as a 5 tuple containing: a set of states (S), a set of actions (A), a transition function (T) that governs how the states evolve, a reward function $R : S \times A \to \mathbb{R}$ that maps states and actions to a reward value, and a discount factor (γ) that governs how much how reward is discounted. The goal of

the MDP is to maximize the future discounted reward collected over time (3.1), and this solution is characterized by a policy ($\pi : S \rightarrow A$).

$$\max \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \tag{3.1}$$

3.2 Approximate Q-Learning

One way to solve an MDP and find a desirable policy is through approximate Q-learning [27]. This method defines a Q-value of a state-action pair $Q(s, a)$ as the expected future discounted reward for selecting action $a \in A$ in the state $s \in S$, serving as a predictor for the reward to be collected. The function Q can be approximated as a linear combination of features of a state-action pair (3.2), with $\boldsymbol{\psi}(s, a)$ as the relevant features of $\langle s, a \rangle$ and $\boldsymbol{\theta}$ as the weight vector, learned through interactions with the environment.

$$Q(s, a) = \boldsymbol{\theta} \cdot \boldsymbol{\psi}(s, a) \tag{3.2}$$

$$\textit{difference} = [r + \max_{a'} Q(s', a')] - Q(s, a) \tag{3.3}$$

$$\boldsymbol{\theta}_i \rightarrow \boldsymbol{\theta}_i + \alpha \cdot \textit{difference} \cdot \boldsymbol{\psi}(s, a) \tag{3.4}$$

For every action a that the agent takes in state s, it obtains an immediate reward r and reaches a future state s'. As the agent takes a sequence of actions over multiple episodes, the weights $\boldsymbol{\theta}$ are updated through (3.3) and (3.4), in which α is the learning rate and γ is the discount factor for future reward.

When it comes to choosing an action that maximizes $Q(s, a)$, the learner often follows and ϵ-greedy approach. With such, a random action will be taken with probability ϵ and an action with maximum Q-value will be taken with probability $(1 - \epsilon)$. This ensures that there is a balance between exploration and exploitation of the Q function.

3.3 Proximal Policy Optimization

Another widely used method to solve MDPs is Actor-Critic Proximal Policy Optimization (PPO) [37]. This method utilizes the concept of advantage of state-action pairs, as well as conservative policy updates in order to stabilize the training procedure. The following subsections cover the technical details on advantage estimation, which is used for training the agent on how to select actions that lead to maximum reward, as well as the policy update procedure.

3.3.1 Generalized Advantage Estimation

When selecting actions in the environment using the MDP, the agent must rely on a method to estimate the expected outcomes of such actions. One of these methods, named Generalized Advantage Estimation (GAE) [38], utilizes the concept of advantage of a state-action pair in order to estimate the outcome of taking an action at a given state. The advantage function (3.5) is the difference between the Q-value of a state-action pair and the value of that state, measuring how much better (or worse) an action is compared to the average of all actions that can be taken. Since the Q-value can be written as the reward plus the discounted value of the next state (3.6), the advantage function can be rewritten (3.7) and expanded (3.8) to model the advantage at an n^{th} state starting from the first state (s_0).

$$\hat{A}(s,a) = \hat{Q}(s,a) - \hat{V}(s) \tag{3.5}$$

$$\hat{Q}(s_0,a_0) = r_0 + \gamma\hat{V}(s_1) \tag{3.6}$$

$$\hat{A}(s,a) = r_0 + \gamma\hat{V}(s_1) - \hat{V}(s_0) \tag{3.7}$$

$$\hat{A}^{(n)}(s,a) = r_0 + \gamma r_1 + ... + \gamma^{n-1} r_{n-1} + \gamma^n \hat{V}(s_n) - \hat{V}(s_0) \tag{3.8}$$

A good way to estimate $A(s,a)$ given recorded trajectories of n steps is to define δ_t as the temporal-difference advantage estimate for time step t (3.9). With that, $A(s,a)$ can be estimated by taking an exponential average over all advantages over the trajectory (3.10), and with expansion and substitution, the equations can be simplified (3.11). Since the advantage estimate depends only on the next-step advantage and the value of δ_t, it is possible to estimate all advantages in the trajectory if the value function (used to compute δ) is known. One common way to model the value function is through a neural network (called critic), fitted over state-reward pairs from recorded trajectories.

$$\delta_t = r(t) + \gamma\hat{V}(t+1) - \hat{V}(t) \tag{3.9}$$

$$\hat{A}_t(s,a) = (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + ...) \tag{3.10}$$

$$\hat{A}_t^{GAE} = \delta_t + (\lambda\gamma)\hat{A}_{t+1}^{GAE} \tag{3.11}$$

3.3.2 Clipped Objective Function

To ensure stable policy evolution towards maximizing rewards, a conservative approach of small policy updates between rollouts is typically adopted. The updates between rollouts are centered on the notion of a policy ratio (3.12), which is the ratio of the probability of taking a certain action a at the state s with the current policy over the probability of a previous policy. Here, the policy is modeled by a neural network (called actor), and θ are the network features. In order to ensure small updates, the policy ratio can be constrained between values of $1-\epsilon$ and $1+\epsilon$ using a clipping function (3.13). This enforces the policy ratio to fall within a trusted region of stability.

$$r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3.12}$$

$$clip(r_t(\theta), 1-\epsilon, 1+\epsilon) \tag{3.13}$$

When applying updates to a policy, the goal is to take a conservative step in the direction maximizing the objective function $L^{CLIP}(\theta)$, defined by the product of the clipped policy ratio with the advantage estimation (3.14). Note that the objective function evaluates to the minimum (lower bound) between the policy ratio itself or the clipped policy ratio. The objective function can be maximized using stochastic gradient descent (SGD), and the argument θ obtained from the maximization is used for the next policy rollout.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{3.14}$$

3.4 IRL with Feature Expectations

Classic reinforcement learning assumes that the reward function is explicitly given to the learner. When agents are learning a policy in the absence of an explicit reward function, it is necessary for the agent to observe a set of trajectories (3.15) executed by an expert, defined as a sequence of states and actions, so that the rewards can be estimated.

$$Tr = (s_0, a_0, s_1, a_1, ..., s_{L-1}, a_{L-1}, s_L) \tag{3.15}$$

In order to model the large space of all possible reward functions, the reward can be represented as a linear combination of $l > 0$ features (3.16). In this model, w_i are the reward weights to be learned, and $\phi_i(s, a, s') \rightarrow \mathbb{R}$ are the reward features of tuple $\langle s, a, s' \rangle$.

$$\hat{\mathbf{R}}_{\boldsymbol{w}}(s, a, s') = \sum_{i=1}^{l} w_i \phi_i(s, a, s') \tag{3.16}$$

The vector projection algorithm [15] utilizes the concept of feature expectations (μ) to evaluate the quality of the learned reward function. Given a set of expert trajectories, the feature expectations $\hat{\mu}_E$ of the executed policy can be empirically defined using (3.17). Here, m represents the number of expert trajectories given, with $\hat{\mu}_E$ being the mean feature expectations over a set of m trajectories.

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{L-1} \gamma^t \boldsymbol{\phi}(s_t^{(i)}, a_t^{(i)}, s_{t+1}^{(i)}) \tag{3.17}$$

When given weights w_i, it is possible to compute $\hat{\mathbf{R}}_{\boldsymbol{w}}$ and estimate $\hat{\mu}(\pi_w)$ in a similar way as (3.17) by training the agent to generate trajectories that maximize the cumulative reward, allowing the algorithm to find the best w_i so that $\hat{\mu}(\pi_w) \approx \hat{\mu}_E$. The algorithm computes the approximation by minimizing $|\hat{\mu}(\pi_w) - \hat{\mu}_E|$ (feature expectation loss) using an iterative vector projection calculation as described in [15].

3.5 Ethical Dilemma Formalism

As defined in Chapter 1, ethical dilemmas are situations in which agents are required to make one of at least two available ethical decisions, and cannot make multiple decisions at the same time [4]. Following this definition, it is possible to construct a mathematical framework that describes the elements that constitute an ethical dilemma. This framework uses set theory to define three sets of elements that are involved in an ethical dilemma.

The $Moral\,Value\,System\,(MVS)$ set is defined as a set of moral values that can guide moral decisions in an ethical dilemma. Individual moral values are represented as elements v_i of the MVS set (Equation 3.18). A moral value system set can be characterized as a collection of moral values that are held by a group of people. For example, the Ten Commandments [39] is a religious moral value system comprised of 10 individual moral values. Another example of a moral value system would be the Doctrine of Double Effect [40], which includes values that permit actions that cause serious harm or death as a side effect of promoting a net positive good.

$$MVS = \{v_1, v_2, ...\}. \tag{3.18}$$

The V set is defined as a set of all moral value systems that can be involved in an ethical dilemma. It serves as a superset that contains all possible moral values that can guide decision-making in the dilemma. Thus, a particular MVS set is a subset of the V set (Equation 3.19). Building from the examples presented in the previous paragraph, the Doctrine of Double Effect and the Ten Commandments would both be included as distinct elements within the V set.

$$V = \{MVS_1, MVS_2, ...\},\ MVS \subseteq V. \tag{3.19}$$

Lastly, the D set is defined as a set of moral decisions that can be made in the dilemma. Individual moral decisions are represented as elements d_i of the D set (Equation 3.20). For example, consider a dilemma scenario in which an agent is faced with the option of killing a kidnapper in order to save a hostage. The two main decisions in this dilemma would be to not kill the kidnapper (d_1) or to kill the kidnapper (d_2).

$$D = \{d_1, d_2, ...\}. \tag{3.20}$$

Based on this framework, we hypothesize that a function $F : V \rightarrow D$ that maps the V domain (set of moral value systems) to the D codomain (set of moral decisions) exists and can be modeled (Figure 3.1). Namely, the function F acts as a moral guide for decisions based on moral value systems at play in a given dilemma. Considering

the previous example of deliberation on whether or not to kill a kidnapper in order to save a hostage, two moral value systems could be involved in the V domain: the Ten Commandments (MVS_1) and the Doctrine of Double Effect (MVS_2). In this dilemma, the function F would map the Ten Commandments (MVS_1) to the decision to not kill (d_1), given that one of the moral values carried by this system is "you shall not murder". On the other hand, the same function would also map the Doctrine of Double Effect moral value system (MVS_2) to the decision to kill (d_2), as killing would be permitted in order to save the hostage. This example illustrates how it would be possible to produce dilemma decisions guided by moral value systems through the function F.

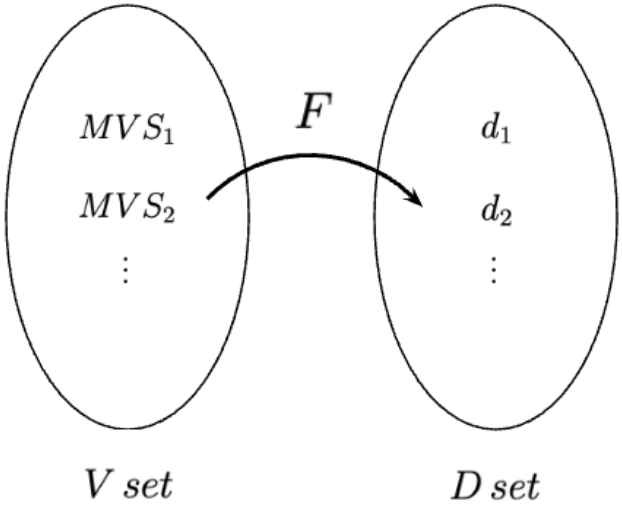

Figure 3.1: Function F maps the V domain to the D codomain.

3.6 Chapter Takeaways

This chapter introduced previously developed methods for training an agent through reinforcement learning, as well as a formalism for modeling decision-making in an ethical dilemma through a mapping between moral value systems and dilemma decisions. Chapters 4 and 5 present experimental applications of these methodologies to train

agents to achieve value alignment in two dilemma scenarios. The architecture presented in Chapter 4 utilizes approximate Q-learning and IRL with Feature Expectations in the learning framework, and Chapter 5's architecture employs the proximal policy optimization technique. Both architectures build upon the ethical dilemma formalism by modeling the function F through the reward function, ultimately mapping demonstrated or codified moral value systems to dilemma decisions.

Chapter 4

Home Invasion: Architecture and Application

This chapter presents details on the architecture and methodology for training an AI agent to make value-aligned decisions on a home invasion dilemma scenario. The first section provides an overview of the dilemma scenario. The second section covers a proposed two-layer architecture that separates decision-making from action execution in the environment. The third section discusses the reward function used and how it maps moral value systems to decisions in the dilemma. The fourth section lays out the implementation details and experimental results.

4.1 The Home Invasion Dilemma Scenario

The home invasion scenario presented in this chapter is centered on the following question: what should an intelligent domestic robot do when it detects an invader breaking into its home? In this scenario, a domestic robot is more capable than a standard home security system because it can move through the environment and perform complex manipulation tasks. The robot's mobility and manipulation capabilities extend the range of decisions that it can possibly make, and some of these decisions may be in moral conflict with

each other.

For example, the robot might follow the castle doctrine moral principle [41] and decide to retrieve a weapon and physically attack the invader. As a second option, it might decide to follow justice-based values and simply call the police and not directly engage with the invader. As a third option, it might be guided solely by self-preservation and decide to hide and not engage. Lastly, it might decide to simply sit idle and do nothing. This scenario constitutes a moral dilemma because the robot is required to make a decision, and each one of these options is guided by a different moral value system.

The home invasion scenario was chosen for this study due to its simplicity. It is relatively simple to model in simulation, as the scenario can unfold in a 2-dimensional floor plan of a home. There is one main decision point, which is the moment when the agent detects the invader. In addition, there are multiple moral value systems that can be used for guiding decision-making, making it adequate for testing an architecture's value alignment capabilities.

4.2 Decision and Execution Policy Architecture

The proposed architecture for the agent to act on the home invasion scenario utilizes two layers: a layer for decision-making and a layer for executing the decision made. The decision-making layer is responsible for handling the high-level consequential decisions in the ethical dilemma situation. In other words, this layer is concerned with what the agent or robot should do when faced with a dilemma. The execution layer is responsible for transforming the decision made by the decision-making layer into actions in the environment. Namely, this layer focuses on how the agent should act in the environment to execute the decision made. The decision-making layer will be called the Decision Policy layer, and the executive layer will be called the Execution Policy layer.

Each layer uses its own Markov Decision Process, with unique state / action spaces and

rewards. The Decision Policy layer only observes state features from the environment that are relevant to the high-level decision-making process, with its action space spanning the different decisions that the agent can make. The Execution Policy layer observes features that allow the agent to move through the environment effectively (such as distances to objects) to achieve a certain objective, and its action space spans motion options (e.g. move forward, move sideways, etc.) allowed by the dynamics of the environment. In the Decision Policy layer, the rewards guide how the high-level decisions should be made, and in the Execution Policy layer the rewards regulate how this motion through the environment should be executed.

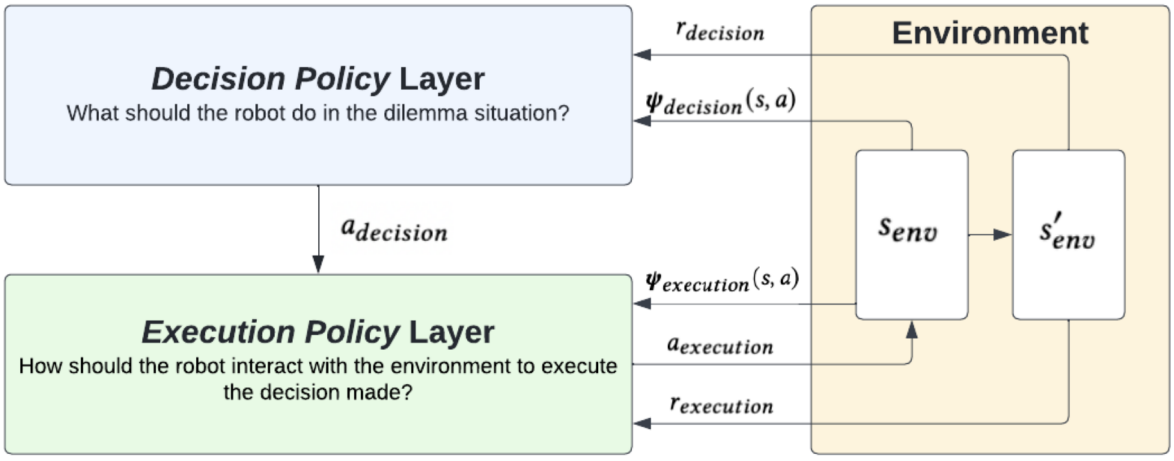

Figure 4.1: Decision and execution policies diagram. Both policies interact with the environment to find the best possible actions.

The two proposed layers follow a hierarchical structure. As shown in Figure 4.1, a decision action ($a_{decision}$) is taken by the Decision Policy layer based on input ($\boldsymbol{\psi}_{decision}$) from the current state of the environment (s_{env}), and this decision is passed on to the Execution Policy layer. This layer will then produce an action ($a_{execution}$) based on $\boldsymbol{\psi}_{execution}$ from s_{env} and, after the action is taken, the environment state evolves to s'_{env}. Lastly, both decision and execution layers collect rewards $r_{decision}$ and $r_{execution}$ from state s'_{env}, and the procedure is repeated until the final outcome is reached.

4.3 Reward Function

In a Markov Decision Process, the reward function is the element that has strongest influence on how policies are shaped for the agent. Within this context, the reward function was used to model the function F from Figure 3.1, mapping moral value systems to decisions in the dilemma.

The reward function that was designed uses features of the state-action pair to learn value systems demonstrated by a human exemplar via inverse reinforcement learning. Those features map different possible end results of a scenario to numerical values that are weighted according to their importance for the value system codified. We make two assumptions: (a) only the most likely end results of the scenario are relevant for modeling the rewards, and (b) a correlation exists between value systems demonstrated by the exemplar and end results that the exemplar reaches in the scenario, with these end results being consequences of the decisions made by the agent.

Expanding on these assumptions, we presume that if a human exemplar demonstrates a specific value system, a corresponding subset of end results will be reached in the scenario based on the exemplar's decisions. With that, among the list of all possible likely end results that can be reached by the agent, the results favored by the demonstrated value system will have higher probability of happening than all other possible results. This probability distribution can be mapped to the reward structure through the reward weights, so that the agent gets rewarded for reaching the results that correlate to the values demonstrated by the exemplar, as shown in Figure 4.2.

Following equation (3.16), the reward function is designed as a linear combination of features. In this model, each of the most likely end results that can be reached by the agent or robot serves as a unique feature $\phi_i(s, a, s')$. All features are Boolean, meaning that if a given end state is reached by the agent, the feature for that state returns a

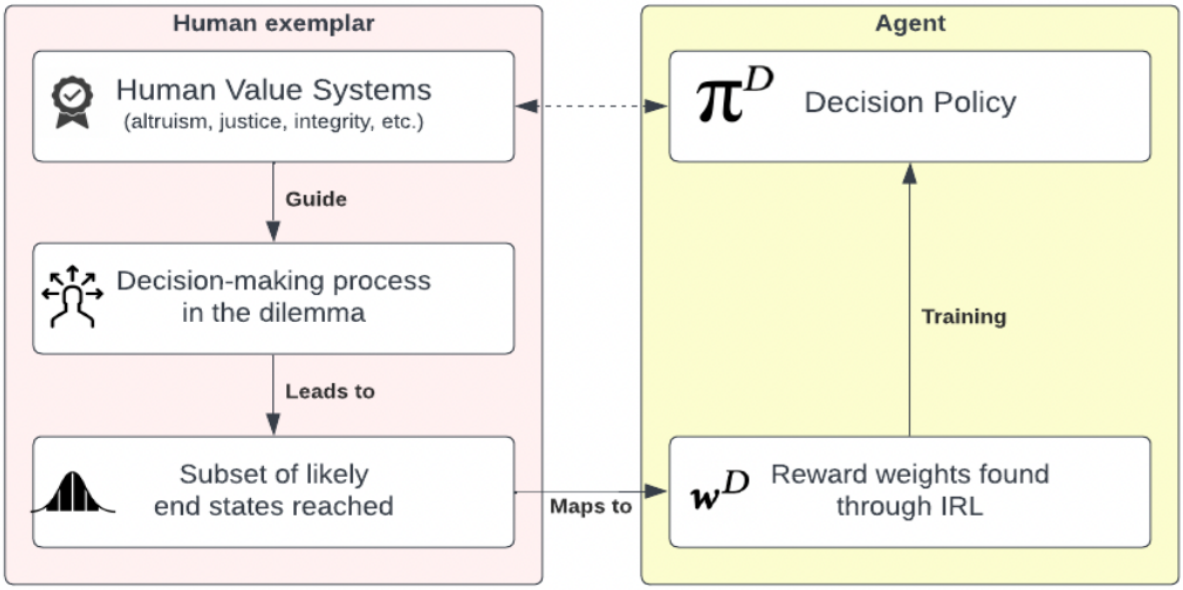

Figure 4.2: Mapping between human value systems and learned policy. Human value systems demonstrated by a human exemplar ultimately feed into a learned policy.

numerical value of 1 while all other features return 0. The weights that multiply each feature are learned from training, and they codify the value system that is demonstrated by the human exemplar.

4.4 Experimental Application

The simulation environment that was used for this study builds from Berkeley AI's Pac-Man game [42], with the inverse reinforcement learning implementation adapted from Noothigattu et al. [26]. The simulation arena (Figure 4.3) represents a house and its surroundings, with the agent placed inside the house and a home invader outside the house. A police officer is also present, and their purpose is to neutralize the intruder if summoned by the agent. The arena includes a weapon that can be used by the agent to kill the intruder. The notional house contains two doors facing a street and one back door that serves as a hiding space.

Our home invasion scenario allows the agent to take actions at two different levels.

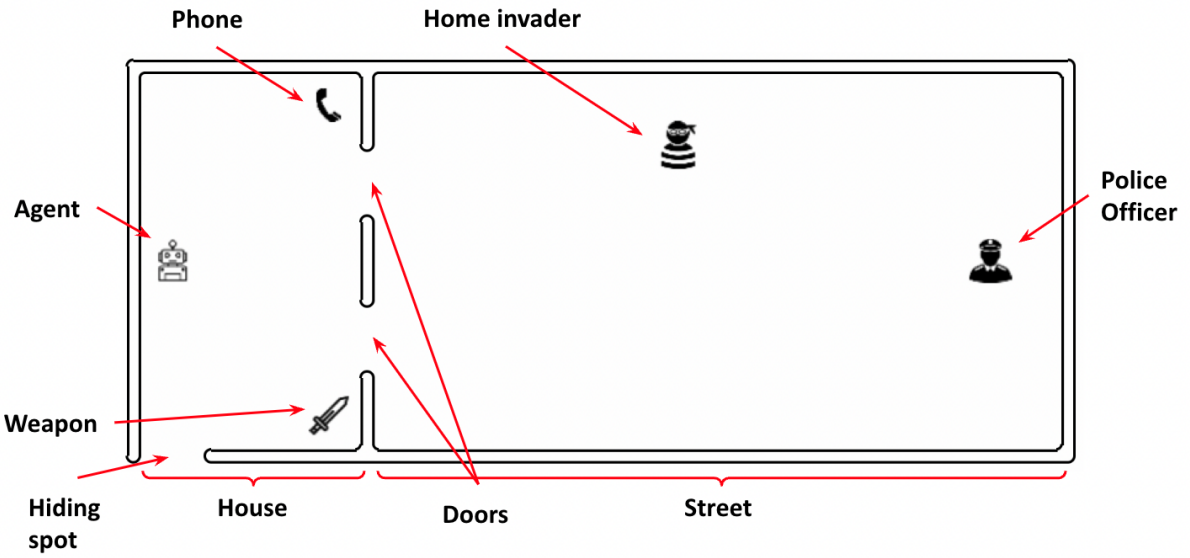

Figure 4.3: The home invasion simulation environment.

At the high-level, the agent makes decisions based on whether an invasion has occurred. At the low-level, the agent determines how to navigate through the environment. The Decision Policy layer controls the high-level decision-making and the Execution Policy layer controls the low-level actions taken in the environment. High-level decision-making starts with the agent in an initial state, representing the moment before the invasion happens. The state then evolves to the invasion itself, which occurs with a 50% probability. Low-level action selection, which is decided by the Execution Policy layer, takes input from the environment's spatial elements including the location of the characters and the items. The simulation begins with the agent inside the house and the invader outside of the house. The subsequent states evolve with a 50% probability that the invader enters the house. The simulation lasts for up to 100 states, with each state corresponding to one time step of the simulation environment.

The low-level actions taken by the agent in the environment lead to the result dictated by the Decision Policy. The low-level action space allows the agent to move in different directions. There are a total of 5 permitted actions: idle, move up, move down, move left and move right. The environment is modeled by a tile grid, so each action allows the

agent to move to an adjacent tile in the following state. Within the high-level action space, the agent is required to take two actions (decisions) within the state space: one action before the invasion happens, and another after the invasion happens. Since the invasion has 50% chance of occurring, in approximately half of simulation episodes the agent does not experience the invasion, so its second action is applied in the context of no invasion. The action space spans 4 possible decisions: remain idle, hide, call the police and kill the invader.

All simulation episodes start from the same initial state, with the agent inside the house and the invader outside, and end when one of the following outcomes is reached: agent dies, invader is killed by the agent, invader is neutralized by the police officer, agent hides, or the simulation times out. If none of the outcomes listed are reached by the 100^{th} state, the episode terminates with a time out outcome. This limit was introduced to speed up the training and testing process, avoid infinite loops, and offer a terminal point in case the invasion does not happen in the episode.

The Q-learning features $\boldsymbol{\psi}(s, a)$ were designed to provide a numerical representation of the state-action pair so that the agent can predict the rewards returned for each action at a given state. For decision policies, the features spanned the four actions the agent is allowed to take plus a state variable tracking whether the invasion has happened. For execution policies, a total of 11 features were used, simulating sensor input from the environment for state tracking. Tables 4.1 and 4.2 include lists of all Q-learning features used in both layers.

4.4.1 Training and Testing Decision Policies

Training for the Decision Policy layer begins with a human experimenter acting as the ethical exemplar and controlling the agent in the simulation in multiple conditions. In the first condition, the experimenter opts for resolving conflict without harm for any of

Table 4.1: Q-learning features $\boldsymbol{\psi}$ and weights $\boldsymbol{\theta}^D$ for decision policies

$\psi_i(s,a)$	$\boldsymbol{\theta}^D_{hide}$	$\boldsymbol{\theta}^D_{call}$	$\boldsymbol{\theta}^D_{kill}$
Bias (always 1)	0.02117	-0.11714	0.25167
Has invasion happened **and** agent not idling?	0.00360	0.05543	0.08734
Has agent chosen to idle?	0.01136	0.00229	0.08734
Has agent chosen to kill invader?	-0.00324	-0.05479	0.06851
Has agent chosen to call police?	0.00438	-0.02184	0.03902
Has agent chosen to hide?	0.00867	-0.04280	0.05760

Table 4.2: Q-learning features $\boldsymbol{\psi}(s,a)$ for execution policies

Index	$\psi_i(s,a)$
0	Bias (always 1)
1	Is invader 1 step away after agent retrieves weapon?*
2	Is invader 1 step away before agent retrieves weapon?*
3	Distance to invader after agent retrieves the weapon*
4	Distance to invader before agent retrieves the weapon*
5	Negative inverse of the distance to phone
6	Negative inverse of the distance to weapon
7	Negative inverse of the distance to the hiding exit
8	Negative inverse of the distance to the police officer
9	Is the agent one step from killing the invader?
10	Has the police officer been called?

* Features based on Noothigattu's implementation [26].

the parts involved. The experimenter takes no action (idles) before the invasion and then hides from the invader once the invasion began. In the second condition, the experimenter chooses to incur punishment for the transgressor without direct engagement. Again, the experimenter takes no action (idles) before the invasion, but called the police after the invasion has started. In the third condition, the experimenter decides to take direct responsibility for their security and protect their property. Here, the experimenter once again takes no action (idles) before the invasion, but then kills the invader after the invasion. In the fourth and final condition, the experimenter demonstrated the case of

Table 4.3: Feature expectations from human exemplar

Feature	$\hat{\mu}_{E_{hide}}$	$\hat{\mu}_{E_{call}}$	$\hat{\mu}_{E_{kill}}$
Bias (always 1)	1.8525	1.8525	1.8525
Result: Police neutralizes invader	0.0	0.45125	0.0
Result: Invader dies	0.0	0.0	0.45125
Result: Agent dies	0.0	0.0	0.0
Result: Agent is a perpetrator	0.0	0.0	0.0

Table 4.4: Reward features ϕ_i and weights $\boldsymbol{w}^D$ for decision policies

$\phi_i(s, a, s')$	$\boldsymbol{w}^D_{hide}$	$\boldsymbol{w}^D_{call}$	$\boldsymbol{w}^D_{kill}$
Bias (always 1)	0.00145	-0.01536	-0.00586
Result: Police neutralizes invader	0.0	0.01175	0.0
Result: Invader dies	-0.00787	0.0	0.02658
Result: Agent dies	-0.00787	-0.02086	-0.02435
Result: Agent is a perpetrator	-0.03357	-0.02926	-0.02503

no invasion, choosing to idle when the invasion does not happen.

Feature expectations ($\hat{\mu}_E$) were measured for each demonstration executed by the human experimenter. Since the final demonstration (no invasion case) is common to all policies, the feature expectations of the other three demonstrations were averaged out with the no invasion case, following equation (3.17). The resulting $\hat{\mu}_E$ are shown on Table 4.3. There, $\hat{\mu}_{E_{hide}}$ corresponds to the objective of hiding after the invasion, $\hat{\mu}_{E_{call}}$ is for calling the police after the invasion, and $\hat{\mu}_{E_{kill}}$ is for killing the invader after the invasion. The value of $\gamma = 0.95$ was used for measurement of $\hat{\mu}_E$ for each policy. The three $\hat{\mu}_E$ values served as input for the IRL projection algorithm, resulting in the reward weights $\boldsymbol{w}^D$ shown in Table 4.4 for decision policies.

These resulting reward weights were fed into the Q-learning algorithm for generating three decision policies (π^D_{hide}, π^D_{call} and π^D_{kill}), with training spanning 30000 episodes. Testing was executed every 100 episodes throughout training, with evaluation of whether

the results reached by the agent matched the results reached by the exemplar. The values of $\gamma = 0.95$, $\alpha = 0.01$ and $\epsilon = 0.1$ (for ϵ-greedy action selection) were used for training the decision policies.

Figure 4.4 shows the most common decisions selected (vertical axis) for each policy when no invasion happened during testing, as the number of training episodes increased from 0 to 30000 (horizontal axis). As shown in the plot, all policies converged to the action of idling after the number of training episodes surpassed 22000. This result means that all three policies achieved the desired objective for that training condition, which was to simply idle when no invasion occurs.

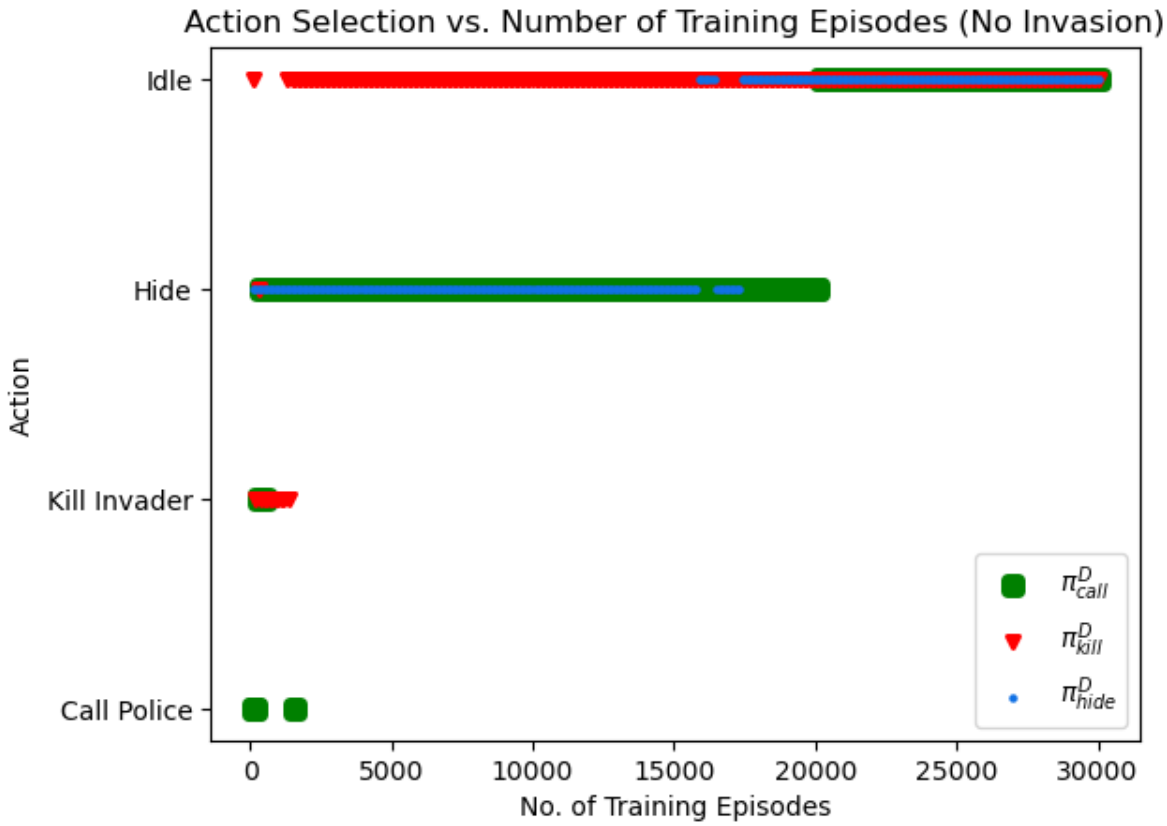

Figure 4.4: Actions selected by each policy throughout training (no invasion case).

For the case of when the invasion happens (Figure 4.5), the desired objectives were met for all three distinct policies when the number of training episodes surpassed 28500. The

π^D_{call} policy took longest to converge, while π^D_{kill} was the quickest. The slow convergence does not pose a practical issue, as an optimal π^D_{call} policy is eventually achieved during training. The plot also shows a "None" option in the bottom, which arises for all three policies. This is due to the training agent choosing actions other than idling before the invasion happened, which preemptively eliminated the possibility of invasion in those episodes.

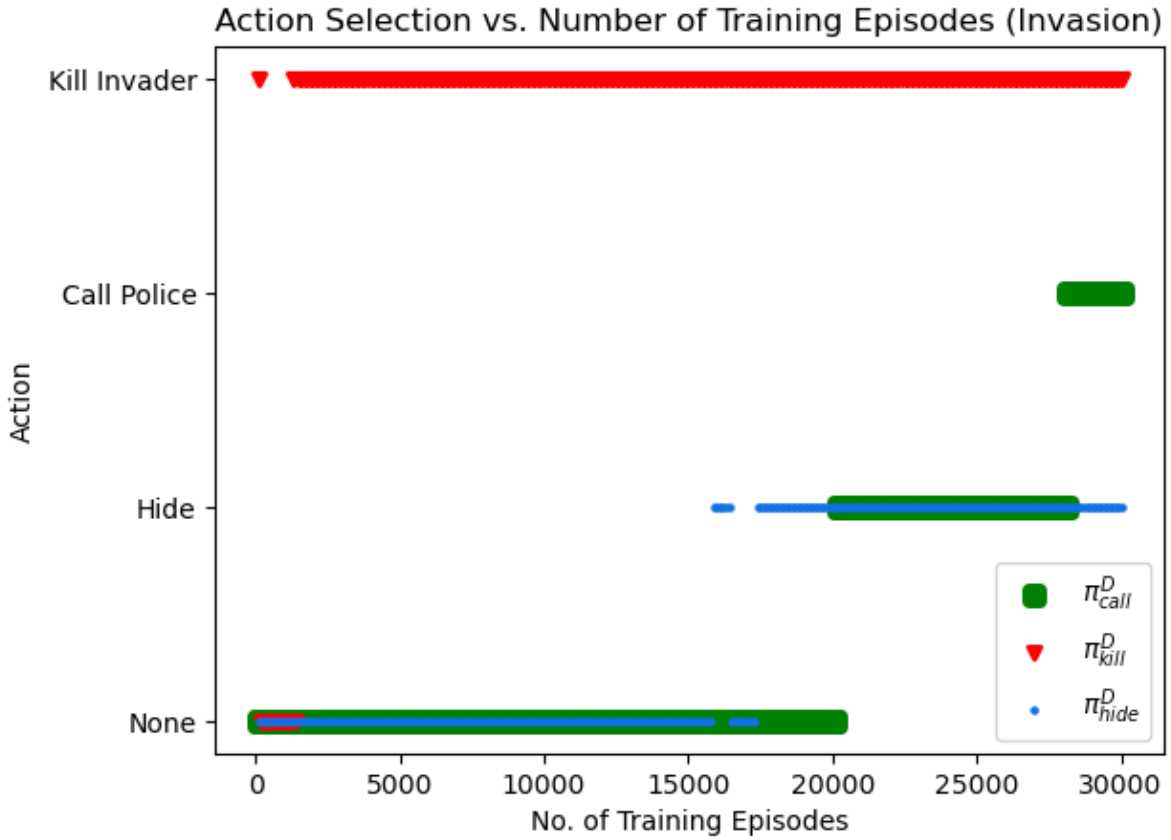

Figure 4.5: Actions selected by each policy throughout training (invasion case).

Following the structure for $\boldsymbol{\psi}(s, a)$ on the decision layer (see Table 4.1), the resulting Q-values for the decision policies are effectively the sum of the weight for the bias term (ψ_0) with the weight for the corresponding decision selected, plus the weight for the conditional term (ψ_1). For example, if the agent chooses to call the police after the invasion, the resulting Q-value of that state-action pair will be θ_0 (bias) + θ_1 (conditional

Table 4.5: Reward features ϕ_i and weights $\boldsymbol{w}^E$ for execution policies

$\phi_i(s, a, s')$	$\boldsymbol{w}^E_{hide}$	$\boldsymbol{w}^E_{call}$	$\boldsymbol{w}^E_{kill}$
Bias	1	1	1
Is the invader killed?	-100	-100	1000
Is the weapon picked up?	-100	-100	100
Is the invader neutralized by police?	-100	1000	-100
Does the agent die?	-100	-100	-100
Does the agent hide?	1000	-100	-100
Does the simulation time out?	-100	-100	-100
Does the agent call the police?	-100	100	-100

term) + θ_4 (call police decision choice).

As shown in the values of $\boldsymbol{\theta}$ (Table 4.1), the highest non-bias dimension (θ_i) in all policies correspond to the choice of idling (θ_2). This means that before the invasion happens ($\psi_1 = 0$), the highest Q-value will always be attributed to the choice of idling. When the invasion happens, on the other hand, the sum of θ_1 with at least one of the values in the set $\{\theta_3, \theta_4, \theta_5\}$ is always greater than θ_2, so the agent chooses not to idle and either hides, calls the police or kills the invader. For π^D_{hide}, $Q_{max} = \theta_1 + \theta_5$; for π^D_{call}, $Q_{max} = \theta_1 + \theta_4$; and for π^D_{kill}, $Q_{max} = \theta_1 + \theta_3$. Based on these solutions, Q_{max} guarantees that the correct decisions are made on all three policies.

4.4.2 Training and Testing Execution Policies

With respect to the execution policy, reward weights were designed so that the agent would learn three different behaviors: move to a hiding spot (π^E_{hide}), move to a phone and summon police (π^E_{call}), and retrieve the weapon and kill the invader (π^E_{kill}). Inverse reinforcement learning was not used on the execution layer because the rewards are well-defined and do not necessarily need to replicate human motion through space. The reward features for each behavior and their respective weights are found in Table 4.5.

For $\boldsymbol{w}^E_{hide}$, the highest weight was set to $w_5 = 1000$, with all other weights (except for

bias term) set to -100. This established an incentive for the agent to reach the hiding location. For $\boldsymbol{w}^E_{call}$, the highest weight was set to $w_3 = 1000$ to reward the agent for getting the invader neutralized by the police, with a second highest weight $w_7 = 100$ to indicate that calling the police is the best method for reaching the desired objective. All other non-bias weights were also set to -100 for $\boldsymbol{w}^E_{call}$. For $\boldsymbol{w}^E_{kill}$, the highest weight was set on $w_1 = 1000$, rewarding the killing of the invader, and a second high weight was set on $w_2 = 100$ to reward the agent for picking up the weapon. Similarly as the other policies, other non-bias weights were set to -100 for $\boldsymbol{w}^E_{kill}$.

The agent learns to execute all three policies with success, generating three sets of Q-learning weights that are callable by the decision-making layer based on interactions with the environment. Training spanned 2000 episodes, and as with the decision policy training, testing was executed every 100 episodes throughout training (train for 100 episodes / pause and test policies / train for 100 more episodes / pause and test policies / ...). Figure 4.6 shows percentage of testing episodes in which the agent successfully reached the end results desired for each policy. In the plot, the policies were tested 30 times in each testing session, with 20 testing sessions executed (600 testing episodes in total). The policy π^E_{hide} achieves fast convergence since it is represented by a single repeated action (move down to hiding location) without need for avoiding collision with the invader, maintaining 100% success as the number of training episodes increased. The policy π^E_{call} presents slightly lower rates because the simulation often reached its 100^{th} frame before the police officer neutralized the invader. For π^E_{call}, the optimal weights were selected at the maximum rate (100%), after the 1600^{th} training episode. The policy π^E_{kill} also achieved fast convergence, with near 100% successful outcomes across most of the span of training. The values of $\gamma = 0.9$, $\alpha = 0.2$ and $\epsilon = 0.1$ (for ϵ-greedy action selection) were applied for training the execution policies.

Figure 4.7 depicts the pathlines of the agent moving within the environment for each

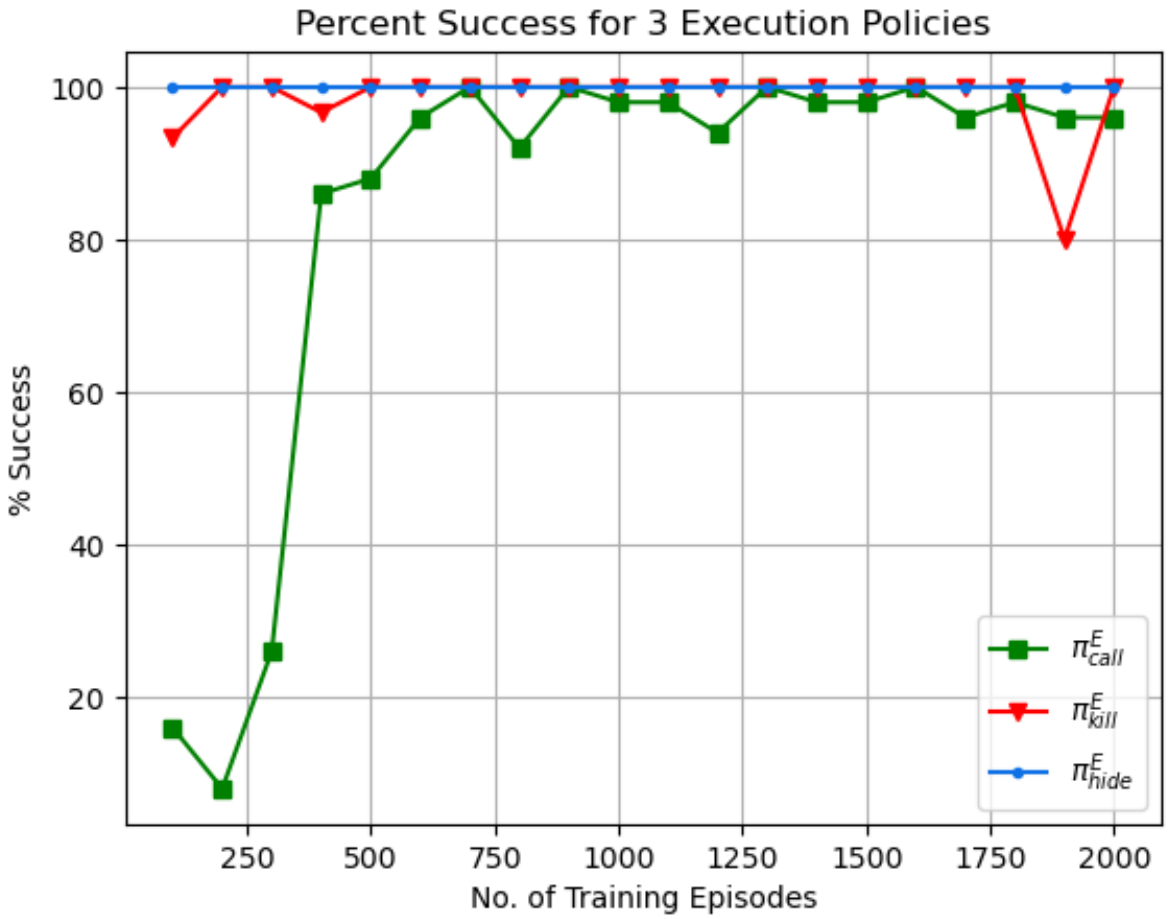

Figure 4.6: Percent success for testing of execution policies throughout training.

policy over 5 tested episodes, with light blue representing the paths for π^E_{hide}, green for π^E_{call} and red for π^E_{kill}. Thicker lines indicate the paths in which the agent passed more often. On π^E_{hide}, the only path the agent took was a straight line from the initial position to the hiding exit. On π^E_{call}, the path shows a pattern of the agent moving up and to the right to reach for the phone icon in order to summon the police. The square loop near the phone location shows that the agent lingers near the phone while it waits for the police officer to arrive inside the house.

For π^E_{kill}, the agent consistently moves to the right and downwards in the direction of the weapon in order to kill the invader. In a particular episode, the agent chose to head outside the house after retrieving the weapon and moved close to the static police officer.

Figure 4.7: Pathlines of execution policies during testing. Each policy was tested graphically five times to generate the paths shown. Blue represents the paths for π^E_{hide}, green for π^E_{call} and red for π^E_{kill}.

The agent waited near the officer for the invader to arrive, and as it reached a position immediately to the left of the officer the agent killed it. I conjecture that the agent chose this path in order to minimize the chance of it being killed by the invader, and it serves as an example of the different types of creative solutions that the reinforcement learning agent can produce to reach its optimal objective.

4.4.3 Robot Demonstration

In order to demonstrate the versatility of the decision policy layer, we arranged a physical environment for demonstration of learned decision policies in a robot. The Pepper robot [43] was used as the agent and a laboratory room as a notional home, with the weapon and phone placed in separate locations within the room (Figure 4.8). The robot was programmed to visually recognize an invader in the room and take an action based on a given learned decision policy. The action of hiding makes the robot move to the back of the room; the action of calling the police makes the robot move to the phone location; and the action of killing the invader makes the robot move to the weapon location.

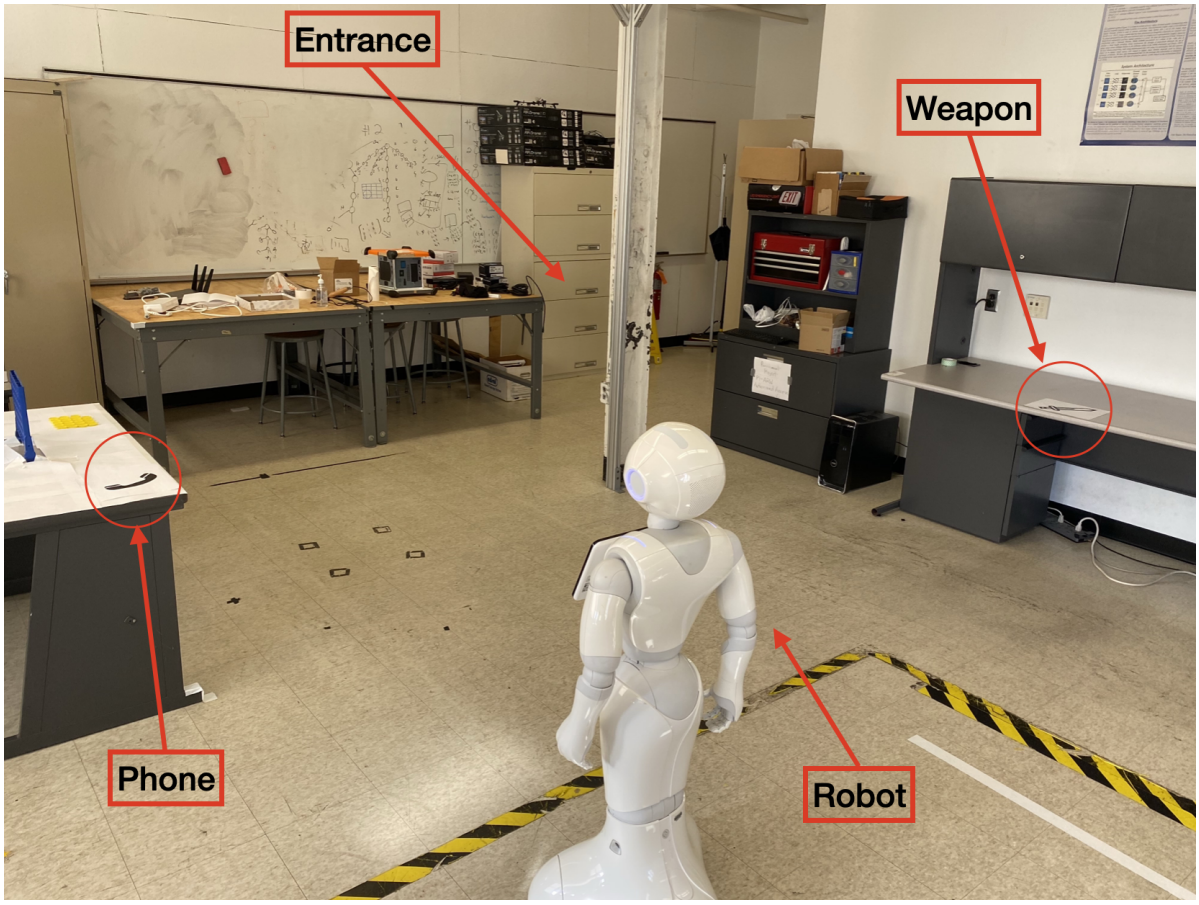

Figure 4.8: Pepper robot in demonstration environment.

It is important to note that no execution policy solutions were used in this robot demonstration. In order to execute the decisions made in the Decision Policy layer, motor control routines were hard-coded for the robot, and these routines were individually called by the Decision Policy layer after a decision was made. The three hard-coded routines consisted of navigation from the robot's initial position to the pre-defined locations of the weapon, phone and hiding spot.

The robot successfully executed all three decision policies in the demonstration environment. The robot was able to recognize when an invader entered the environment in all three policy demonstrations, and execute the three different motor control routines for notionally hiding, calling the police, and killing the invader. This demonstration proves that even though the policies are trained in a virtual simulation, they can be

applied to a physical domain with a real robot and maintain the same level of success.

4.4.4 Lessons Learned

In this chapter, we present an architecture that segments the learning process into a Decision Policy layer and an Execution Policy layer, with a reward structure that allows the agent to learn value systems demonstrated by a human exemplar and apply the learned policies on a dilemma scenario. This architecture was implemented on a home invasion scenario, successfully showcasing that the agent is capable of learning distinct value-guided ethical behaviors using inverse reinforcement learning. The agent reproduced three value-guided decision policies in the simulation and in the robot demonstration: hide after invasion, call the police after invasion and kill the invader after invasion. Reinforcement learning was also leveraged to teach the agent to interact with the simulation environment in order to execute the decisions made, generating three execution policies that reached the objectives desired by the three trained decision policies. With these results, We have demonstrated that the architecture can make decisions that are aligned to dominant moral value systems at play.

Despite its success, this architecture carries a few concerns. The first concern is that the reward function relies on a set of finite scenario end results. It would be difficult to compile a list of relevant scenario end results for a real-world home invasion case, in which there are potentially infinite outcomes that can be achieved by both the robot's and invader's actions. A second concern regards the necessity of training the agent in two separate levels. If this architecture was scaled to a scenario that presents a large number of decision options, the total number of policies to be trained would be double the number of decision policies attainable, representing a scalability issue. In Chapter 5 we aim to address these concerns by introducing an architecture which does not rely on scenario end results or separation between decision-making and action execution.

Chapter 5

Search and Rescue: Architecture and Application

This chapter details the architecture and methodology for training an AI agent to make value-aligned decisions in a search and rescue dilemma scenario. The first section provides an overview of the dilemma scenario. The second section covers the environment setup, including details on the framework used for modeling the state space. The third section discusses the reward architecture used and how it serves as a model for function F, mapping moral value systems to decisions in the dilemma. The fourth section lays out the implementation details and experimental results.

5.1 The Search and Rescue Dilemma Scenario

The search and rescue scenario presented in this chapter involves an agent tasked with rescuing multiple hostages from a threat actor. The agent's mission is to rescue all hostages, and as it executes its mission it must decide whether or not to kill the threat actor. Different moral value systems could point to different preferred decisions in this scenario, as some may view killing as acceptable in this case while others may view it as always unacceptable. Similarly to the home invasion case, this scenario constitutes a

moral dilemma because the agent must make a decision, and both decision options are in moral conflict with each other.

The architecture proposed in this chapter differs from the dual-layer architecture introduced in Chapter 4 in a few key ways. First, the goal of this architecture is to achieve value alignment without separation of execution policies from decision policies. With this architecture, we aim to explore how an agent can associate low-level actions in the environment directly with abstract moral decisions, without the need of a distinct arbitration layer for decision-making. For this search and rescue scenario, a single layer of execution is trained to produce actions in the environment, and those actions must be aligned with a top-level abstract moral directive that allows or denies killing the threat actor. This approach potentially addresses the scalability concern raised in the previous chapter, since it only requires a single policy to be trained for each dilemma decision considered.

Another key difference between this chapter's architecture and the previous chapter's is the reward function structure. The search and rescue reward features are based on a few relevant events that happen throughout the scenario, and not the scenario's end results. With that, the reward function for this search and rescue architecture enables scalability of end results that can be achieved by the agent, representing an advantage over the previous chapter's architecture.

The algorithm of choice for training the agent in this search and rescue scenario was Proximal Policy Optimization (PPO). This algorithm introduces complexity to the training process when comparing to the Q-learning technique used in the previous chapter, but its powerful pattern recognition capability allows the agent to develop more intricate abstractions over low-level states and actions. This added capability is a key enabler for this architecture to map low-level action selection to abstract moral decisions.

5.2 Environment Setup

The Pac-Man environment developed by Berkeley AI is also used for training the agent in this search and rescue scenario. The training arena consists of a rectangle of 18-by-9 tiles, equivalent to the classic Pac-Man arena dimensions. The scenario includes Pac-Man repurposed as the AI agent; 8 food elements representing hostages to be rescued by the agent; 5 walls that serve as obstacles and 4 walls that provide the boundaries to the arena; one ghost representing a threat actor; and one Power Pellet representing a weapon that allows Pac-Man to kill the threat actor. A typical episode starts with the agent placed at the center of the arena, the threat actor placed at the top left-hand corner, the hostages scattered across the arena, and the weapon placed at the bottom right-hand corner. The threat actor is programmed to move towards the agent upon start of the episode, and the agent is tasked with rescuing as many hostages as possible. The positions of the 5 obstacle walls, 8 hostages, weapon, as well as starting positions of the agent and threat actor were fixed for all training episodes.

The episode length was limited to 100 steps to prevent infinite loops in the agent's behavior. The episode is set to end early if the threat actor touches the unarmed agent, simulating the threat actor killing the agent. If the agent retrieves the weapon and touches the threat actor, the threat actor's speed is set to 0 and it is relocated to its original position in the top left-hand corner, simulating the agent killing the threat actor. The episode also ends as soon as the agent rescues all 8 hostages.

The agent is allowed to take 4 different actions in this scenario: up, down, left or right. When the agent's location is adjacent to a wall or obstacle, the actions that lead the agent to move in the direction of the wall or obstacle are deemed illegal. If the agent chooses to take an illegal action, the result is equivalent to the agent choosing to remain static and not move. For example, if there is a wall to the left of the agent and the agent

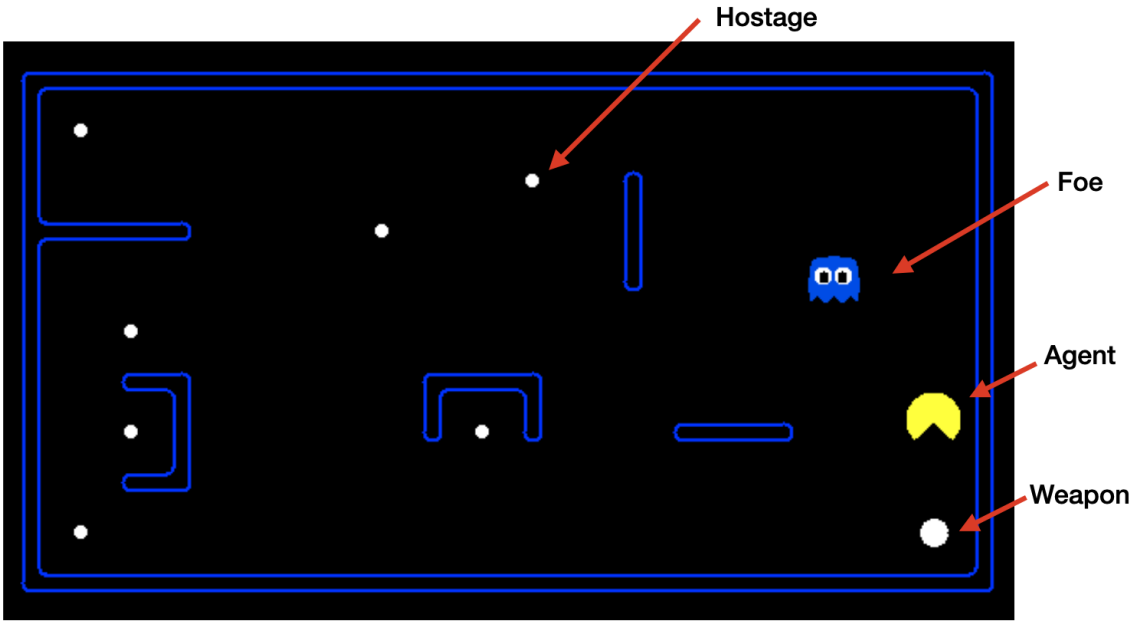

Figure 5.1: Arena setup for search and rescue scenario.

selects the action of moving left, in the next state the agent remains in the same place and does not move.

5.2.1 State Vector Modeling

The state vector represents how the agent perceives the environment. The state vector was modeled as a 9-tile matrix grid (Figure 5.2) surrounding the agent. The agent is located at the center of the grid, with 8 tiles covering the adjacent space. The tiles are indexed column-wise from bottom-to-top and left-to-right, with the bottom-left tile receiving index 0, middle-left tile with index 1, and so on. Each tile in the matrix is modeled as key-value pair, with the key being the tile's index, and the value being a numerical representation of the elements of the scenario that may be in the tile.

The tile grid is fixed to the agent's frame of reference, hence, as the agent moves through the environment the grid moves with it. When the agent nears different elements of the environment, such as hostages or the threat actor, the tiles in which those elements

are located assume specific numerical values. An empty tile assumes the value of 0; the walls assume the value of 1; the threat actor takes the value of 2; the weapon takes the value of 3; and the hostages are assigned the value of 4. This numerical attribution simulates a robot perception system that reads the immediate surroundings of the agent and encodes the reading into a 9-column vector that can be easily ingested by the learning algorithm. Since there can be up to 5 different values attributed to each tile in the matrix, and as there are 8 tiles surrounding the agent, there are a total of $5^8(390625)$ possible states comprising the state space.

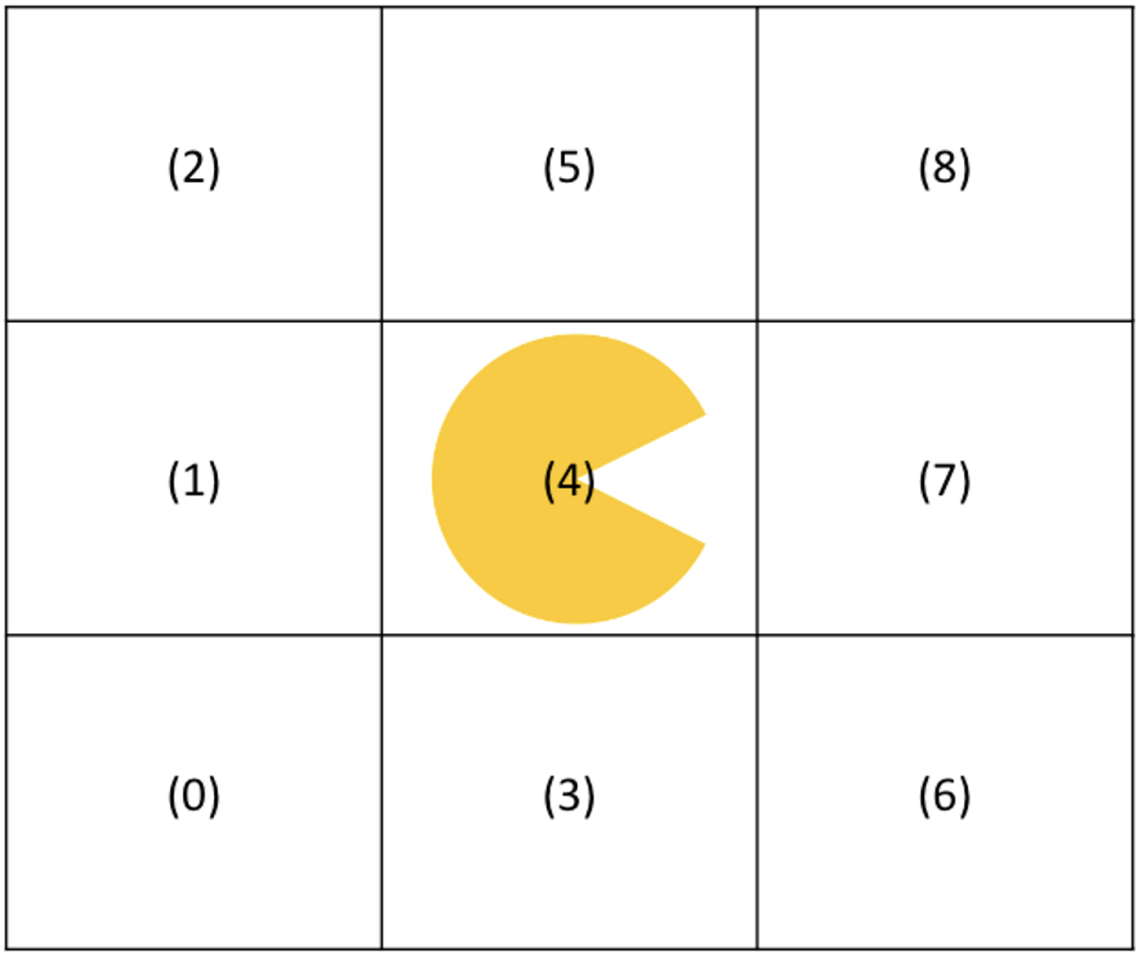

Figure 5.2: Tile grid simulating robot perception system.

Table 5.1: Designed numerical rewards

Condition / Dimension	Reward
Agent rescues hostage	+25
Agent kills threat actor	-10
Agent takes a step	-0.02
Agent is killed by threat actor	-50
Agent takes illegal step	-5

5.3 Reward Function Design

The reward function maps moral value systems to dilemma decisions through careful selection of numerical values that reward or penalize the agent for specific scenario outcomes. For this implementation, a moral value system that prohibits the agent from killing the threat actor was used as a basis for crafting the reward features and values. With that, a successfully value-aligned agent would be expected to execute the search and rescue mission without killing the threat actor.

The rewards were chosen to provide an incentive for the agent to rescue hostages, while penalizing it for killing the threat actor, getting killed by the threat actor, or taking an illegal step. The agent is also penalized for taking each individual step, so that there is an incentive for completing the search and rescue mission in the least number of steps possible. The numerical rewards for each condition are displayed in Table 5.1. Those numerical values were selected based on trial and error, with an initial guess established for the reward distribution and subsequent updates made to increase the policy performance. Several reward distributions were tested, with different scales and magnitude of differences between the reward dimensions, and the rewards that led to highest number of hostages rescued on average were selected for the winning policy.

Table 5.2: Designed training parameters

Parameter	Value
Learning rate	2.5×10^{-4}
Learning rate annealing	False
Number of steps per policy rollout	128
Number of training environments	-50
Batch size	512
Number of minibatches	4
Minibatch size	128
Discount factor (γ)	0.99
Clipping Coefficient (ϵ)	0.2

5.3.1 Training Procedure

Proximal policy optimization training was executed over a total of 400,000 parallelized steps, with 4 environments running in parallel (totaling 1.6 million training steps). This parallelization was done to speed up the training process and facilitate training of the critic network for value estimation. Training parameters were selected based on previous PPO implementations for solutions of toy problems [44]. A complete list of parameters is displayed on Table 5.2.

Learning rate annealing was not used for training because the tested performance of the agent with a constant learning rate was higher than with an annealed learning rate. Minibatch stochastic gradient descent with adaptive momentum (Minibatch SGD / Adam) [45] was the algorithm of choice for objective function maximization. The discount factor γ was set as 0.99 in order to balance out long-term rewards with short-term gains.

The actor and critic networks were modeled as Multilayer Perceptrons (MLPs), with an input layer, one hidden layer, and one output layer. The hidden layer's width was set to 64 nodes, with hyperbolic tangent (tanh) activation function. Both critic and actor networks took the 9-value flattened out state vector grid as inputs. The critic

network's output corresponds to the value estimation for the state input, whereas the actor network's output corresponds to the action to be taken by the agent based on the state input.

The plots on Figure 5.3 display some of the training performance metrics tracked. The entropy plot measures how random the decisions of the actor network are made. A combination of decreasing entropy as training progresses, along with convergence to a non-zero value, indicates a healthy training run. The explained variance plot measures how good the critic network is at predicting the value of a given state. Positive explained variance together with convergence above 0.3 represents successful training.

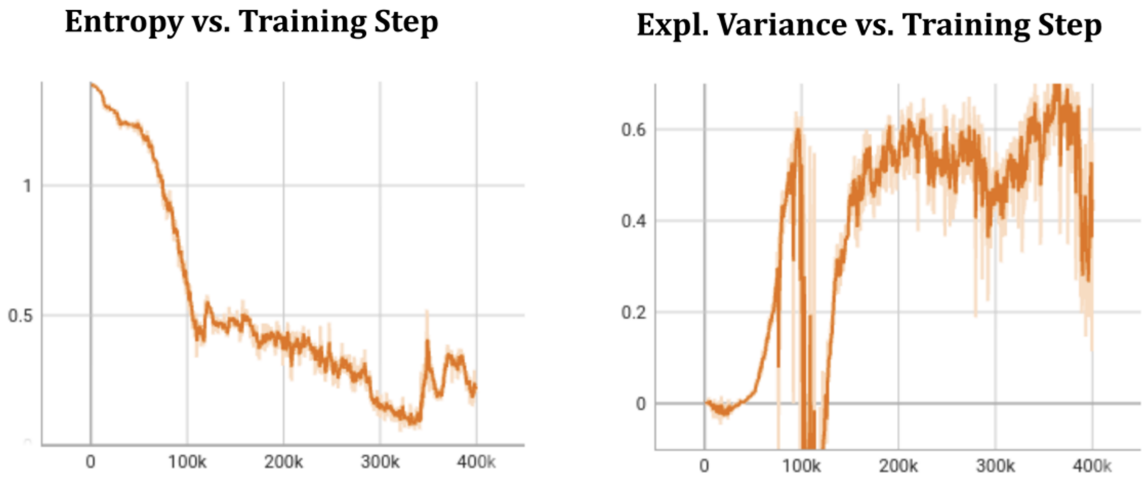

Figure 5.3: Plots of training metrics: entropy and explained variance.

5.4 Results

The learned policy was tested across 100 episodes, at first with the same scenario configuration used for training. During testing, the agent demonstrated that it learned to maximize the number of rescued hostages, with the average of number of rescued hostages close to 8. A list of test metrics and the resulting performance is displayed in

Table 5.3: Testing performance over unmodified scenario configuration

Performance Metric	Value
Average number of rescued hostages	7.83
Percentage of episodes in which the agent kills the threat actor	76%
Percentage of episodes in which the agent rescues all hostages	86%
Average number of illegal actions per episode	0.4
Average number of steps per episode	65.9

Table 5.3. These metrics served as the main performance measures for the learned policy, highlighting key behavior patterns developed by the agent.

As shown in Table 5.3, the agent rescues all 8 hostages in 86% of testing episodes, with a total average of number of rescued hostages per episode very close to 8. The agent also learns to avoid taking illegal steps, averaging less than 1 illegal step per episode. The average number of steps per episode is less than the maximum number of steps the agent can take in the scenario, indicating that the agent learns to rescue all agents in the least number of steps possible.

One key result is that the agent kills the threat actor in 76% percent of testing episodes, even though the reward function penalizes killing the threat actor. This result could potentially mean that the penalty for killing the threat actor is not strong enough. With that in mind, one potential solution for preventing the agent from killing the threat actor is to tune the reward function by increasing the magnitude of the negative reward associated with this outcome. The agent was further trained with different reward magnitudes for killing the threat actor, ranging from -11 to -20 in increments of -1. Each policy was tested using the first two metrics shown on Table 5.3, with the goal of finding a policy that maintains a high number of rescued hostages while reducing the incidence of killing the threat actor. A plot of the tracked metrics across the range of reward values used is presented below in Figure 5.4.

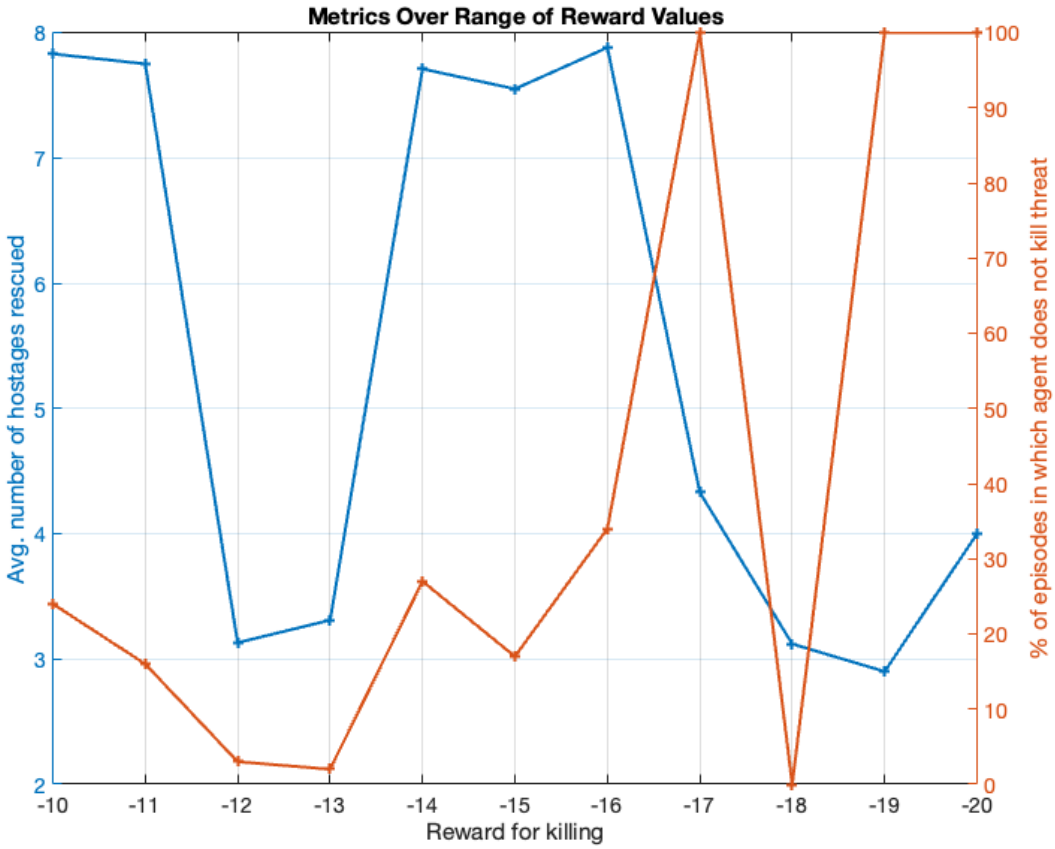

Figure 5.4: Performance metrics variation across range of reward values.

The blue line plots the average number of rescued hostages during testing for each reward value, and the brown line plots the percentage of testing episodes in which the agent did not kill the threat actor. The left-most points in the plot represent the original trained policy with reward of -10 for killing the threat actor. The chart shows wide variation as the magnitude of the negative reward increases from -10 to -20, with no attainable trend for maintaining high number of rescued hostages while reducing threat actor kills. As the negative magnitude of reward increases, the percentage of episodes in which there is no kill increases, as desired, but the rescue performance decreases. The reward of -16 yields a high rescue performance with slightly higher percentage of no threat actor kills than the original policy, but this policy is still distant from the desired

target of 100% no threat actor kills. On the right-hand region of the plot, as the reward value approaches -20, the percentage of no threat actor kills reaches the desired target of 100%, but that comes at a cost of lowering the number of rescued hostages to an average of 4, which is half of the desired target.

There are two dominating objectives at play in this scenario: one is to maximize the number of hostages rescued, and the other is to not kill the threat actor. The results displayed in Figure 5.4 demonstrate that the agent is not able to meet both objectives simultaneously. When the penalty for killing the threat actor increases, the agent trades rescuing more hostages for not killing the threat actor. When the penalty for killing is lower, the agent incurs the cost of killing in order to rescue more hostages. When looking at these results from a moral values standpoint, the system's behavior indicates that under these experimental conditions it is only possible to achieve value alignment (i.e. not kill the threat actor) at the expense of reducing the total number of rescued hostages.

In order to evaluate generalizability of the learned policy and gain further understanding of the solution, the policy was tested in a series of modified scenario configurations. The configuration variations covered the availability and location of the weapon element, number and location of hostages, behavior of the threat actor, and number of obstacles. The subsections below contain testing summaries for these modified configurations.

5.4.1 Weapon Configuration

The policy was tested in environments with modified weapon configurations in order to evaluate if the policy was agnostic to weapon location and availability. The modifications entailed making the weapon inaccessible by the agent by walling off the bottom right-hand corner, as well as placing the weapon in different locations within the arena. Table 5.4 contains the performance metrics for the various tested configurations, again with 100 episodes used in each test.

Table 5.4: Testing performance over modified weapon configurations

Metric	Weapon inaccessible	Weapon in top-right corner	Weapon in bottom-left corner	Weapon near top wall
Average number of rescued hostages	2.98	2.96	2.98	4.26
% of episodes in which the threat actor is killed	0%	0%	0%	82%
% of episodes in which all hostages are rescued	0%	0%	0%	0%
Average number of illegal actions per episode	4.44	4.17	5.27	1.11
Average number of steps per episode	28.74	30.71	32.03	79.89

As shown in Table 5.4, the learned policy is very sensitive to weapon availability. When the weapon is inaccessible to the agent, the number of hostages rescued is significantly lower than the in the original configuration, and the agent never rescues all hostages. The number of illegal actions increases dramatically, since the agent moves to the weapon location and tries to force its way to access it through the wall, without success. The average number of steps is much lower than in the original configuration, since the agent is prematurely killed by the threat actor in all tested episodes.

The tested policy also presents lower performance when the weapon is not located in the bottom-right corner. When the weapon is available at the top-right or bottom-left corners, the agent never retrieves it, and the number of rescued hostages is low. This shows that the learned policy did not learn to search for the weapon, instead it memorized the weapon location in the bottom-right corner during training. When the weapon is placed near the agent in the top wall, the agent retrieves it and kills the threat actor in 82% of the testing episodes, but the number of hostages rescued is still low and the agent never finishes rescuing all hostages. In summary, these results indicate that weapon

Table 5.5: Testing performance over modified hostage configurations

Metric	12 hostages to be rescued	16 hostages to be rescued	8 hostages / different locations (1)	8 hostages / different locations (2)
Average number of rescued hostages	9.51	11.60	4.82	5.02
% of episodes in which the threat actor is killed	95%	96%	98%	100%
% of episodes in which all hostages are rescued	0%	0%	0%	0%
Average number of illegal actions per episode	0.96	0.92	1.22	1.03
Average number of steps per episode	96.69	96.85	98.58	100

location is crucial for overall success of the policy, and variations in weapon location lead to lower performance.

5.4.2 Hostage Configuration

The next variation explored different locations and numbers of hostages in the scenario. This time the policy was tested in an environment with more hostages to be rescued, as well as the same number of hostages located in different positions in the environment. The testing results are presented in 5.5.

When more hostages are present in the environment, the learned policy rescues a higher number of hostages than the original setup with 8 hostages, indicating that it is not sensitive to the number of hostages available. With respect to killing the threat actor, the policy tends to kill more when there are more hostages available, with over 95% and 96% of episodes in which the threat actor is killed over configurations of 12 and 16 hostages, respectively. The agent also tends to use more steps to accomplish the mission on average, nearing the limit of 100 steps in both configurations. It is also worth

noting that the agent did not rescue all of the hostages in any of the different hostage configurations.

When keeping the number of hostages constant but varying their locations in the environment, the number of hostages rescued decreases significantly. Hostages were placed in random locations across the arena for 2 testing runs, and in both runs the policy rescued around 5 hostages on average, never finishing the mission. In both runs the percentage of episodes in which the threat actor was killed neared 100%, along with the average number of steps nearing 100. This is an indicator that the policy is conditioned to the positions of the hostages in the arena, with variations in the positions leading to lower performance.

5.4.3 Threat Actor Configuration

A third configuration variation examined how the threat actor moves in the environment. This variation considered two situations: one in which the threat actor is present in the environment but does not move, and the other in which the threat actor moves away from the agent when the agent possesses the weapon. A summary of the testing results is presented in 5.6.

Table 5.6 indicates that the policy performes well with respect to number of rescued agents, and the number of threat actor kills are low. This result makes sense because in both testing cases the chances of the agent not finishing the rescue mission are significantly reduced. The main takeaway from this result is that the agent is not conditioned to necessarily kill the threat actor. The performance metrics are not negatively affected by the absence of the threat actor, and when the threat actor is set to move away from the agent, the number of kills is significantly reduced. This suggests that the agent is not chasing the threat actor in order to kill it, the killings happen mainly because the threat actor gets in the way of the agent throughout the mission.

Table 5.6: Testing performance over modified threat actor behavior

Metric	Static threat actor	Threat actor moves away
Average number of rescued hostages	7.88	7.74
% of episodes in which the threat actor is killed	0%	1%
% of episodes in which all hostages are rescued	92%	84%
Average number of illegal actions per episode	0.38	0.36
Average number of steps per episode	63.42	66.20

One metric that is not tracked here is the percentage of episodes in which the agent retrieves the weapon. When visually inspecting the policy over this scenario variation, the agent consistently moves to the weapon location in the bottom-right corner regardless of the behavior of the threat actor. This reinforces the idea that the agent is conditioned to move to weapon location, as the paths traced by the agent inevitably pass through the bottom-right corner.

5.4.4 Obstacle Configuration

The last variation considers the number of obstacle walls present in the environment. There was a total of 3 configurations tested: no obstacles, three obstacles, and seven obstacles (note that the original policy was trained with 5 obstacles present). Table 5.7 summarizes the results for these configurations.

When there are fewer obstacles present than in the trained configuration, the policy achieves lower performance in terms of number of rescued hostages. The policy also presents higher threat actor kill rates, at 100% in both cases of zero and three obstacles

Table 5.7: Testing performance over modified obstacle wall configurations

Metric	No obstacles	3 obstacles	7 obstacles
Average number of rescued hostages	5.26	5.60	0.03
% of episodes in which the threat actor is killed	100%	100%	1%
% of episodes in which all hostages are rescued	0%	0%	0%
Average number of illegal actions per episode	0.27	0.71	16.90
Average number of steps per episode	100	100	22.69

present. The agent never finished the mission in both cases and used all 100 steps available in all episodes. This result suggests that the policy is very sensitive to the number of obstacles present, overfitting its solution to the condition of 5 obstacles present.

When there are seven obstacles present, all performance metrics are significantly affected. The average number of rescued hostages nears zero, the average number of steps taken is low, and the average number of illegal actions is very high. With this condition, the agent tries to follow its memorized path to rescue the hostages, and as it encounters unforeseen obstacles, it gets stuck trying to move past them by choosing illegal actions. This makes the agent vulnerable, and the threat actor easily kills it as a result. This very poor performance offers supporting evidence for the idea that variations in the obstacle configuration lead to lower performance.

5.5 Lessons Learned

As discussed in the previous section, the architecture raises a trade-off between maximizing the number of rescued hostages and not killing the threat actor. Since it cannot meet

both objectives at the same time, value alignment comes at the cost of performance in the search and rescue task in terms of number of rescued hostages. This result reflects how the reward function was designed, as the resulting policy is obtained from maximization of rewards. One possible way to remove the trade-off would be to perform robust grid search over the reward space in order to find a solution that maximizes the number of rescued hostages while maintaining value alignment by lowering the number of threat actor kills. While this approach may lead to a policy that meets both value alignment and task performance objectives, finding a solution would be computationally expensive, time-intensive, and not guaranteed, since the reward space is comprised of 5 dimensions.

Looking back at this thesis' research question, the results support the idea that reward function design is one of the most important factors at play when designing architectures capable of value alignment. Reward values strongly influence how final policies are shaped, and reward tuning is crucial for finding solutions that meet performance and alignment objectives. Given the criticality of the reward function, it is important to choose robust design methods that guarantee success of mission objectives along with value alignment. For the case of this experiment, manually tuning the reward function and performing small-scale grid search was not sufficient for finding a solution that leads to value alignment while maintaining high performance in the search and rescue task.

In terms of the results of generalizability testing, one can interpret that the learned policy can be categorized as a path plan based on the training environment. The policy is highly conditioned to specific positions of elements in the environment, such as the weapon, obstacle and hostage positions, representing a memorized path that achieves the mission objectives in most testing episodes. The policy's performance decreases when the environment is subject to element positioning modifications, and the path traced by the agent is mostly unchanged even though the environment is modified. This leads to conclusion that the policy does not solve the problem of "what to do" in the scenario,

instead it solves the problem of "where to go" based on memorized element locations.

When it comes to possible avenues for improving the solution's generalizability, one clear candidate is randomizing the locations of the weapon, obstacles and hostages during training. This can potentially lead the agent not to memorize a path through the environment and instead perform a true search for hostages and weapon, since their positions will be unpredictable. This could make the policy more robust and generalizable, but there is no guarantee that the same reward distribution used here will lead to a converging solution, so additional reward search / tuning may be required.

Chapter 6
Conclusion

6.1 Summary

The objective of this thesis was to evaluate how to design a system that uses reinforcement learning to produce value-aligned decisions in an ethical dilemma. A novel formalism for decision-making on an ethical dilemma was developed, modeling the dilemma structure through three sets of elements and a mapping function: a $Moral\,Value\,System\,(MVS)$ set, a V set and a D set, with a function F mapping moral value systems to decisions. This formalism served as a foundation for two reinforcement learning architectures that were applied to two different ethical dilemma scenarios: a home invasion scenario and a search and rescue scenario.

The two proposed architectures used different reinforcement learning techniques to model the function F and achieve value alignment. The home invasion architecture attempted to achieve value alignment using observation of the decision made by a human ethical exemplar and by using an architecture that uses a separate decision-making layer and low-level action execution layer. The search and rescue architecture aimed to achieve value alignment by codifying a moral value system directly in the reward function through handcrafted reward features and weights, without observing a human ethical exemplar.

The home invasion architecture aimed to achieve value alignment through direct reproduction of decisions made by an ethical exemplar in the ethical dilemma. A total of three moral value systems were demonstrated by the ethical exemplar to produce three different decisions in the home invasion scenario: resolve conflict without harm (decision of hiding); incur punishment for the transgressor without direct engagement (decision of calling the police); and property protection at all costs (decision of killing). Using inverse reinforcement learning with the decision-making layer was successfully able to generate three sets of reward weights, one for each moral value system, leading to three distinct high-level decision-making policies for the agent: the policy of hiding (π^D_{hide}), the policy of calling the police (π^D_{call}), and the policy of killing the invader (π^D_{kill}). The decision policies were complemented by execution policies trained through pure reinforcement learning, leading the agent to achieve high marks of value alignment for all three moral value systems at play.

The search and rescue architecture's goal was to achieve value alignment by directly codifying a moral value system in the reward function to produce a specific dilemma decision. The reward function was designed to discourage the learning agent from killing the threat actor, by punishing it during training through a negative reward associated with the act of killing. The resulting policy demonstrated that the agent chose to kill the threat actor in 76% of test runs in the unmodified scenario, despite the negative reward for killing. A range of negative reward values was applied throughout multiple training runs, with results showing that the overall performance on the search and rescue task decreases as the reward for killing gets more negative. The policy that achieved the highest number of rescued agents was tested against variations in environment configurations in order to evaluate generalizability of the solution, with the results indicating that small variations in scenario configurations lead to significant decrease in performance.

6.2 Contributions

This work explores how an intelligent, autonomous agent might make a difficult decision when faced with an ethical dilemma. The formalism for decision-making presented on Chapter 3 serves as a novel contribution to the field of machine ethics, defining a framework for building techniques that can extract moral values involved in a scenario and produce dilemma decisions that are value-aligned. Due to the relativistic nature of value alignment, it is paramount for artificially intelligent agents to consider moral value systems at play in a moral dilemma and to map them to decisions that are aligned with what humans would do in the scenario.

This formalism was used as a model for two distinct reinforcement learning architectures, with the home invasion architecture presenting better value alignment than the search and rescue architecture according to the defined success criteria. Even though the home invasion architecture was demonstrated to work, it heavily relied on end results, or outcomes, of the scenario that were identified and hand-picked. This design may reduce the architecture's scalability, because, in more complicated scenarios, it might be infeasible to map all potential end results to the reward function. Another potential issue is the small decision space considered. In more complicated scenarios there could be a large number of potential decisions that the agent can make, so constraining the agent to a small set of decisions might lead the agent to disregard potentially value-aligned decision options.

One critical concern raised around the home invasion architecture is its reliance on demonstrations of value-aligned behavior by a human exemplar. Since the human exemplar is responsible for training the agent, if the exemplar demonstrates bad or immoral behavior that is not locally value-aligned, the agent will replicate this behavior, perhaps causing additional ethical problems. Furthermore, because agents are trained

locally by individual human exemplars, these agents will individualistically follow their own moral code based in their training, making it difficult or impossible for an observer to predict or explain the agent's behavior without knowing how the agent was trained. These concerns could be addressed by introducing exogenous constraints on the agent's policies dictated by ethical theories and principles, as proposed by previous work in this field [25, 26, 36].

With respect to the search and rescue experiments, the main limitation emerged from the difficulty of tuning a reward function to produce a value-aligned policy without relying on inverse reinforcement learning. Due to the large multidimensional space of numerical reward values, searching for a set of values that achieves the objective of maximizing the number of rescued hostages without killing the threat actor was difficult. When comparing these results with those demonstrated in Chapter 4, we conclude that inverse reinforcement learning may serve as a better design approach for achieving value alignment, as it bypasses the process of searching for reward values.

In terms of policy generalizability, which is defined as the ability to maintain performance even over conditions not experienced during training, the search and rescue architecture also demonstrated limitations. The results from testing the trained policy on modified environment configurations pointed to a significant decrease in performance, indicating that the training policy was overfitted to the training configuration. A potential solution to this problem would be to diversify the training set by implementing variations in environment configurations, as discussed in section 5.5 of Chapter 5.

6.3 Future Work

The ethical dilemma formalism introduced in this thesis opens up several avenues for further development of architectures that will allow agents to make value-aligned decisions. While this work only explored the use of reinforcement learning for modeling the function

F, other technologies such as transformer models [46] must also be evaluated as potential methods for mapping moral value systems to dilemma decisions. The ethical dilemma formalism is foundational, technology-agnostic and future-proof, so as new machine learning methods and technologies continue to be developed and improved, they must be considered as candidates for architecture implementation.

Generalizability is the Holy Grail of machine learning research and development. It represents the discovery of patterns that enable the solution of problems in scenarios that go beyond those exposed during the system's training period. Consequently, future architectures must be developed with the goal of mapping moral values to dilemma decisions in a way that works for all scenarios and their variations. This means that a system must be capable of extrapolating from its narrow set of training conditions in order to handle a wide range of dilemma situations. It is impossible to create training conditions that expose agents to all combinations of moral value systems, environments, and decision options that ethical dilemmas can involve, so it is crucial for agents to have the extrapolation capabilities that will allow them to handle unseen scenarios.

There are multiple ways through which generalizability can be addressed in future architectures. One way would be to focus on a family of scenarios that involve a specific domain. For example, a home security robot could be trained to incorporate moral value systems prevailing in a household to produce value-aligned decisions in any dilemma scenarios that involve matters of home security for that household. Another focus point would be the transferability of learned morality between different domains. As an example, an agent trained to follow the moral imperative of "you shall not murder" in a home invasion scenario would follow the same imperative when facing the untrained scenario of search and rescue against a threat actor.

As one of the most consequential technologies ever created, artificial intelligence has an immeasurable potential for impacting humanity. With the increase in responsibilities

assumed by autonomous agents in society, it is crucial for these agents to be value-aligned with us, humans, so that this technology is beneficial instead of detrimental. Perhaps the work presented in this thesis will enable a path for development of value-aligned robots and virtual agents, serving as an opportunity for benign and constructive coexistence of artificial intelligence with human life.

Bibliography

[1] S. Russell, D. Dewey, and M. Tegmark, "Research priorities for robust and beneficial artificial intelligence," *Ai Magazine*, vol. 36, no. 4, pp. 105–114, 2015.

[2] C. Bartneck, C. Lütge, A. Wagner, and S. Welsh, *An introduction to ethics in robotics and AI*. Springer Nature, 2021.

[3] I. Gabriel, "Artificial intelligence, values, and alignment," *Minds and machines*, vol. 30, no. 3, pp. 411–437, 2020.

[4] T. McConnell, "Moral Dilemmas," in *The Stanford Encyclopedia of Philosophy*, 2022nd ed., E. N. Zalta and U. Nodelman, Eds. Metaphysics Research Lab, Stanford University, 2022.

[5] B. Duignan, "Trolley problem," in *Encyclopedia Britannica*, 2021.

[6] Ø. Kvalnes, *Moral Dilemmas*. Cham: Springer International Publishing, 2019, pp. 11–19. [Online]. Available: https://doi.org/10.1007/978-3-030-15191-1_2

[7] "Bombing of dresden," Jul 2023. [Online]. Available: https://www.britannica.com/event/bombing-of-Dresden

[8] E. Lipton, "A.I. Brings the Robot Wingman to Aerial Combat," *The New York Times*, 2023. [Online]. Available: https://www.nytimes.com/2023/08/27/us/politics/ai-air-force.html

[9] N. Soares and B. Fallenstein, "Aligning superintelligence with human interests: A technical research agenda," *Machine Intelligence Research Institute (MIRI) technical report*, vol. 8, 2014.

[10] G. M. Briggs and M. Scheutz, ""sorry, i can't do that": Developing mechanisms to appropriately reject directives in human-robot interactions," in *2015 AAAI fall symposium series*, 2015.

[11] J. F. Horty, *Agency and deontic logic*. Oxford University Press, 2001.

[12] K. Arkoudas, S. Bringsjord, and P. Bello, "Toward ethical robots via mechanized deontic logic," in *AAAI fall symposium on machine ethics*. The AAAI Press Menlo Park, CA, USA, 2005, pp. 17–23.

[13] R. Arkin, "Governing lethal behavior: Embedding ethics in a hybrid deliberative/reactive robot architecture. part 3: Representational and architectural considerations," 2007.

[14] S. Bringsjord, K. Arkoudas, and P. Bello, "Toward a general logicist methodology for engineering ethically correct robots," *IEEE Intelligent Systems*, vol. 21, no. 4, pp. 38–44, 2006.

[15] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[16] L. Jiang, J. D. Hwang, C. Bhagavatula, R. L. Bras, M. Forbes, J. Borchardt, J. Liang, O. Etzioni, M. Sap, and Y. Choi, "Delphi: Towards machine ethics and norms," *arXiv preprint arXiv:2110.07574*, 2021.

[17] D. Hendrycks, C. Burns, S. Basart, A. Critch, J. Li, D. Song, and J. Steinhardt, "Aligning ai with shared human values," *arXiv preprint arXiv:2008.02275*, 2020.

[18] M. Anderson and S. L. Anderson, *Machine ethics*. Cambridge University Press, 2011.

[19] P. Lin, K. Abney, and G. A. Bekey, *Robot ethics: the ethical and social implications of robotics*. MIT press, 2014.

[20] W. Iba and P. Langley, "Exploring moral reasoning in a cognitive architecture," in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 33, no. 33, 2011.

[21] R. C. Arkin, P. Ulam, and B. Duncan, "An ethical governor for constraining lethal action in an autonomous system," Georgia Institute of Technology, Tech. Rep., 2009.

[22] S. Tolmeijer, M. Kneer, C. Sarasua, M. Christen, and A. Bernstein, "Implementations in machine ethics: A survey," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–38, 2020.

[23] D. Abel, J. MacGlashan, and M. L. Littman, "Reinforcement learning as a framework for ethical decision making," in *Workshops at the thirtieth AAAI conference on artificial intelligence*, 2016.

[24] Y.-H. Wu and S.-D. Lin, "A low-cost ethics shaping approach for designing reinforcement learning agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[25] A. Balakrishnan, D. Bouneffouf, N. Mattei, and F. Rossi, "Using multi-armed bandits to learn ethical priorities for online ai systems," *IBM Journal of Research and Development*, vol. 63, no. 4/5, pp. 1–1, 2019.

[26] R. Noothigattu, D. Bouneffouf, N. Mattei, R. Chandra, P. Madan, K. R. Varshney, M. Campbell, M. Singh, and F. Rossi, "Teaching ai agents ethical values using reinforcement learning and policy orchestration," *IBM Journal of Research and Development*, vol. 63, no. 4/5, pp. 2–1, 2019.

[27] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[28] S. Russell, *Human compatible: Artificial intelligence and the problem of control*. Penguin, 2019.

[29] V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, R. Kumar, Z. Kenton, J. Leike, and S. Legg, "Specification gaming: the flip side of AI ingenuity," *DeepMind Research*, 2020. [Online]. Available: https://www.deepmind.com/blog/specification-gaming-the-flip-side-of-ai-ingenuity

[30] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, p. 103500, 2021.

[31] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[32] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning." in *IJCAI*, vol. 7, 2007, pp. 2586–2591.

[33] E. Klein, M. Geist, B. Piot, and O. Pietquin, "Inverse reinforcement learning through structured classification," *Advances in neural information processing systems*, vol. 25, 2012.

[34] E. Klein, B. Piot, M. Geist, and O. Pietquin, "A cascaded supervised learning approach to inverse reinforcement learning," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 1–16.

[35] M. Rodriguez-Soto, M. Lopez-Sanchez, and J. A. Rodriguez-Aguilar, "Multi-objective reinforcement learning for designing ethical environments." in *IJCAI*, 2021, pp. 545–551.

[36] A. Loreggia, N. Mattei, F. Rossi, and K. B. Venable, "Preferences and ethical principles in decision making," in *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, 2018, pp. 222–222.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2018.

[39] T. E. o. E. Britannica, "Ten Commandments," in *Encyclopedia Britannica*, 2023.

[40] A. McIntyre, "Doctrine of Double Effect," in *The Stanford Encyclopedia of Philosophy*, 2023rd ed., E. N. Zalta and U. Nodelman, Eds. Metaphysics Research Lab, Stanford University, 2023.

[41] M. Metych, "Stand-your-ground laws," 2023. [Online]. Available: https://www.britannica.com/topic/stand-your-ground-laws#ref1309533

[42] J. DeNero and D. Klein, "Teaching introductory artificial intelligence with pac-man," in *First AAAI Symposium on Educational Advances in Artificial Intelligence*, 2010.

[43] A. K. Pandey and R. Gelin, "A mass-produced sociable humanoid robot: Pepper: The first machine of its kind," *IEEE Robotics & Automation Magazine*, vol. 25, no. 3, pp. 40–48, 2018.

[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.