NORTHWESTERN UNIVERSITY

Responsible Defense from Multi-Drone Attacks

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Tonmoay Deb

EVANSTON, ILLINOIS

March 2026

ABSTRACT

Responsible Defense from Multi-Drone Attacks

Tonmoay Deb

Commercial drones weaponized by non-state actors achieve attack success rates above 70%; yet existing counter-drone systems focus on detection, rather than threat assessment and legally compliant response. This dissertation develops four systems for autonomous urban drone defense.

DEWS (Drone Early Warning System) addresses the threat prediction problem by classifying drone trajectories as threatening or benign based on initial flight segments. The system integrates trajectory dynamics, drone capabilities, no-fly zone violations, and asset-value mappings to enable threat assessment within operationally relevant time-frames. Evaluation on 349 real-world trajectories provided by the Dutch Police over eight months in The Hague demonstrates that DEWS achieves F1-scores exceeding 0.80 within 30 seconds of initial observation for high-threat classification. Performance improves with longer observation windows, reaching precision of 0.967 and recall of 0.869 at six minutes. Feature analysis reveals that asset-based features constitute the most predictive category,

indicating that threat assessment depends fundamentally on what infrastructure a drone threatens rather than trajectory characteristics alone.

STATE (Safe and Threatening Adversarial Trajectory Engine) addresses the scarcity of threatening trajectory data through conditional generative adversarial networks that synthesize trajectories based on geographic context and threat intent. The architecture employs a threat alignment network that enforces consistency between generated trajectories and intended threat classes. When evaluated by law enforcement experts from The Hague on trajectories generated for previously unseen regions, STATE achieves F1-scores of 0.888 for safe and 0.709 for threatening trajectories, representing improvements of 3.62% and 35.8% respectively over variational autoencoder baselines. The system enables security agencies to generate training data for regions lacking operational drone tracking infrastructure.

POSS (Pareto-Optimal Status Sets) provides formal foundations for multi-objective decision-making under legal constraints. The framework combines deontic logic to represent legal norms with Pareto optimization to identify action sets that are both legally compliant and operationally efficient. POSS employs two-stage processing: first pruning actions that violate legal constraints, then computing Pareto-optimal subsets from remaining actions. While initially validated on autonomous vehicle scenarios, the framework's principles apply directly to drone defense contexts requiring similar tradeoffs between operational effectiveness and regulatory compliance.

GUARDIAN (Governance-Unified Aerial Reinforcement-Defense In Accordance with Norms) demonstrates the practical integration of legal and/or ethical constraints with

reinforcement learning for multi-agent drone coordination. The system combines POSS-based constraint satisfaction with QMIX reinforcement learning to coordinate defensive drone swarms. Experimental evaluation across varying grid sizes and drone ratios reveals that incorporating legal constraints during training requires increased computation time (630 hours versus 80 hours for 64 drones) compared to non-constrained training, but maintains operational viability with decision times averaging 625.7 milliseconds. Results indicate that compliance requirements can improve defensive performance in scenarios where defenders are outnumbered, suggesting that constraints help focus policy exploration on viable action spaces. These findings challenge assumptions that legal compliance necessarily compromises tactical effectiveness in autonomous defense systems.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Dr. V.S. Subrahmanian, for his unwavering support and guidance throughout my doctoral journey. His mentorship extended far beyond academic advising; he provided the flexibility and encouragement that allowed me to pursue industry experience through internships, which significantly enriched my research perspective and see the bigger picture of applicability. His open mindedness toward balancing academic rigor with practical experience has been instrumental in shaping both my research and career trajectory.

I extend my sincere appreciation to my dissertation committee members: Dr. Larry Birnbaum, Dr. Nabil Alshurafa, and Dr. Alberto Quattrini Li, who served as the external committee member. Their insightful feedback, challenging questions, and constructive criticism during my proposal and defense were invaluable in strengthening this work.

This dissertation would not have been possible without the contributions of numerous collaborators across multiple institutions. I am profoundly grateful to Dr. Mingi Jeong from Dartmouth College. Mingi was instrumental in helping building the DUCK testbed.

I owe a special debt of gratitude to Dr. Cristian Molinaro from the University of Calabria, who significantly contributed to my understanding and development of the POSS framework. His patience in answering my numerous queries and his willingness to engage in lengthy discussions were essential to the success of that work. I also thank Dr. Andrea Pugliese for his collaborative spirit and technical expertise on our joint projects.

I am grateful to Dr. Eugene Santos Jr. from Dartmouth College, Dr. Youzhi Zhang from the Chinese Academy of Sciences, and Dolev Mutzari and Dr. Sarit Kraus from Bar Ilan University for their valuable contributions to our collaborative research.

I wish to extend special thanks to Dr. Valerio La Gatta and Dr. Marco Postiglione, whose contributions to this dissertation were extraordinary. Their tireless efforts in facilitating the collaboration with our government partners in the Netherlands were indispensable to the success of the DEWS and STATE projects. They played a pivotal role in coordinating with the Netherlands Police and the Municipality of The Hague, supporting the data collection process, and contributing significantly to the preparation and writing of our papers. Beyond their technical contributions, their friendliness, helpfulness, and collaborative spirit made working with them a genuine pleasure.

I extend my heartfelt thanks to our collaborators in the Netherlands who made the DEWS and STATE projects possible. From the Netherlands Police, I thank Sven de Laaf and Odette Lemmens for their partnership and domain expertise. From the Municipality of The Hague, I am grateful to Max van Meerten and Afke Neeleman for their contributions. I also acknowledge Dr. Roy Lindelauf and Herwin Meerveld from the Netherlands Defence Academy for their valuable input and collaboration.

I am deeply thankful to my lab mates, particularly Lirika Sola and Natalia Denisenko, for their camaraderie, intellectual discussions, and support throughout the ups and downs of doctoral research. I also extend my gratitude to Dr. Saurabh Kumar, who was a postdoctoral scholar in our lab before joining IIT Hyderabad. His encouragement and guidance during the early stages of my doctoral journey helped me develop a clear vision for planning and navigating the path toward completing my Ph.D.

I extend my appreciation to the Department of Computer Science at Northwestern University and the Buffett Institute of Global Affairs for their administrative support and resources throughout my doctoral program. I am also grateful to the faculty members for whom I served as a teaching assistant; their flexibility with my research commitments allowed me to balance my teaching responsibilities with my scholarly pursuits.

I would like to offer special thanks to my roommates, Sayak Chakrabarty and Dr. Imon Banerjee, for their friendship and support during this journey. Sayak provided valuable perspectives on industry careers that helped me think about my professional future, while Imon offered thoughtful feedback on my dissertation proposal and defense that proved extremely helpful during those critical milestones.

I would be remiss not to acknowledge those who first inspired my passion for research during my undergraduate years in Bangladesh. I am deeply grateful to Dr. Mohammad Rashedur Rahman and Adnan Firoze for their early mentorship and encouragement to pursue research. I also extend my sincere thanks to Dr. Amin Ahsan Ali for fostering my research interests during my time as his Research Associate.

Finally, and most importantly, I express my deepest gratitude to my family. To my father, Alak Ranjan Deb, for his patience, encouragement, and emotional support throughout the journey. To my mother, Shima Rani Shome, for her love, sacrifices, and constant prayers for my success. To my wife, Arpita Datta, for standing by my side through the struggles and triumphs of this journey. Her understanding, patience, and encouragement during the long nights and stressful periods were immeasurable gifts. My family's support have been the foundation upon which all of my achievements rest.

Dedication

In loving memory of my late grandparents

Nemai Charan Dey *and* **Dipti Rani Debi**

I wish you were here to witness this moment.

This achievement is as much yours as it is mine.

Table of Contents

References 178

List of Tables

List of Figures

CHAPTER 1

Introduction

The world is increasingly populated by unmanned aerial vehicles (UAVs), commonly known as drones. Originally developed for military reconnaissance and operations, drones have rapidly transitioned into commercial, civilian, and recreational sectors, becoming integral to industries ranging from logistics and agriculture to disaster response, entertainment, and real estate (101; 96). Major corporations such as Amazon, UPS, and Walmart have invested heavily in drone delivery systems (125), while emergency services increasingly rely on aerial surveillance to assess disaster zones and coordinate rescue efforts. Insurance companies deploy drones to inspect properties and assess claims (124), and sports arenas use them to capture dynamic footage of events. This widespread adoption reflects the practical benefits drones offer: cost-effectiveness, versatility, and access to locations difficult for humans to reach.

This technological revolution simultaneously introduces unprecedented security vulnerabilities. Off-the-shelf drones provide attack vectors that malicious actors exploit with devastating effect. The dual-use nature of drone technology means that the same platforms employed for benign commercial purposes can be readily weaponized or used for malicious surveillance, making the challenge of distinguishing threats from legitimate operations particularly acute.

This dissertation addresses a critical challenge at the intersection of artificial intelligence, cyber-physical systems, and urban security: how to proactively defend densely

populated urban areas from hostile drone activities while maintaining legal and ethical compliance.

1.1. Motivation

The widespread availability of drones has created a dangerous asymmetry. The same technology enabling innovative commercial applications can be weaponized with minimal technical expertise. This dual-use challenge manifests clearly in the actions of armed non-state actors, such as ISIS (3), the PKK (108), Hezbollah (57), and Lashkar-e-Taiba[1], who have weaponized commercial drones for reconnaissance, targeted strikes, infrastructure disruption, and psychological warfare operations.

The operational effectiveness of drone-based attacks is disturbingly high. ISIS fielded drone swarms during the battle for Mosul (106). Hezbollah has conducted numerous reconnaissance missions over Israeli territory using off-the-shelf unmanned systems (13), while the PKK has utilized drones in attacks against Turkish security forces (108). The 2019 drone attacks on Saudi oil refineries[2] demonstrated that even critical infrastructure with sophisticated security systems remains vulnerable. Documented evidence indicates that 76 drone attacks occurred globally before 2019, achieving a success rate exceeding 70% (5).

[1] https://www.indiatoday.in/india/story/drone-attack-initial-probe-lashkar-role-jammu-and-kashmir-police-chief-1820679-2021-06-29
[2] https://www.nytimes.com/2019/09/14/world/middleeast/saudi-arabia-refineries-drone-attack.html

This threat has intensified dramatically in recent years[3]. The May 2025 India-Pakistan conflict[4] and the ongoing conflict in Ukraine[5] have witnessed extensive use of drones for offensive operations and urban attacks, demonstrating that hostile drone activities represent an immediate and evolving challenge.

Modern consumer drones amplify this threat. Today's commercial platforms offer flight times exceeding 30 minutes (18), operational ranges of several kilometers (49), payload capacities sufficient for explosive devices (31), and autonomous navigation capabilities requiring minimal operator skill (76). As technology advances, the barrier to entry for drone-based attacks continues to diminish.

Even as drone interception techniques on sparsely populated regions improved recently[6], urban environments remain particularly challenging. Cities are characterized by dense populations, critical infrastructure, high-value assets, and complex three-dimensional spaces providing cover and concealment (15). A single weaponized drone penetrating urban airspace can threaten government buildings, transportation hubs, power infrastructure, or public gatherings within minutes. The physical density and vertical complexity of urban terrain provide adversaries with numerous approach vectors while constraining defenders' ability to employ aggressive countermeasures without risking collateral damage.

The legitimate expansion of commercial drone operations in urban airspace compounds this challenge. As package delivery (125), insurance assessments (124), and aerial photography become routine, defenders face an increasingly difficult signal-detection problem:

[3] https://www.youtube.com/watch?v=DfCspYYPi5I
[4] https://www.bbc.com/news/articles/cwy6w6507wqo
[5] https://www.understandingwar.org/backgrounder/russian-drone-innovations-are-likely-achieving-effects-battlefield-air-interdiction
[6] https://united24media.com/latest-news/ukraine-begins-mass-production-of-30-kilometer-fiber-optic-drone-reel-8032

distinguishing malicious drones from benign operations. Legal and regulatory frameworks governing urban drone operations remain fragmented across jurisdictions and often lag technological capabilities (131). This regulatory complexity, combined with the imperative to avoid false positives, renders the urban counter-drone problem distinctly challenging.

Technical approaches for countering UAVs have concentrated predominantly on detection and tracking[7]. Radio frequency detection systems[8] identify drone communication signatures (13), radar systems track small aerial objects (61), and computer vision systems identify drones optically (71). While these detection modalities have matured, they typically provide only binary presence information rather than assessing hostile intent or threat severity. As defense experts at the Modern War Institute emphasize, "The earlier you detect a threat, the sooner you can alert the force while air defense operators work to defeat the threat."[9]

Detection alone is insufficient. Security agencies must distinguish threatening drones from benign operations within seconds of detection, while managing dual risks: false positives (disrupting legitimate activities) and false negatives (resulting in catastrophic security failures). The critical challenge lies not merely in knowing a drone is present, but in rapidly determining whether it poses a threat and formulating an appropriate response. This assessment must occur within compressed timelines, often less than 5 minutes from detection to potential impact (46).

[7] https://www.twz.com/air/ukraines-acoustic-drone-detection-network-eyed-by-u-s-as-low-cost-air-defense-option
[8] https://www.zvook.tech/en
[9] https://mwi.westpoint.edu/understanding-the-counterdrone-fight-insights-from-combat-in-iraq-and-syria/

Existing counter-drone systems rarely incorporate the legal, regulatory, and ethical constraints that must govern defensive actions in civilian environments. International humanitarian law (126; 42), domestic aviation regulations (47), rules of engagement (60), and collateral damage concerns impose strict limitations on permissible responses. An autonomous defense system prioritizing effectiveness while disregarding these constraints is potentially counterproductive, risking civilian casualties, legal liability, and erosion of public trust. Defensive systems must answer not only "Can we stop this threat?" but also "How can we stop this threat?"

The scarcity of comprehensive drone trajectory data compounds these technical and normative challenges. Privacy regulations, operational security concerns, and nascent monitoring infrastructure mean security agencies possess limited historical data about both benign and threatening flights (59). This data scarcity constrains developing automated response, limits adversarial tactic exploration through simulation, and hinders defensive system validation. The few threatening drone flights that have been documented provide insufficient training data for robust predictive models.

These converging challenges define the problem space this dissertation addresses: operational drone threats from state and non-state actors, the urban signal-detection problem, technical gaps in threat assessment, legal and ethical constraints on responses, and data scarcity. When a drone appears in urban airspace, security personnel must answer urgent questions:

- Is this drone's flight pattern threatening or benign?
- What are its likely targets?
- What defensive actions are both effective and legally permissible?

- How can we train autonomous systems to make these determinations reliably?
- How can we test defensive strategies when real-world data on hostile flights is scarce?

1.2. Problem Statement and Research Agenda

We now elaborate on the questions posed in the previous section. The central problem motivating this dissertation is: *How can defenders rapidly identify threatening drone trajectories in urban airspace, formulate legally compliant defensive responses, adaptively learn effective counter-strategies, and overcome data scarcity limitations, all while operating under severe time constraints and ensuring actions satisfy legal and ethical norms?*

This overarching problem decomposes into 4 interrelated research agenda items:

Research Agenda 1: Early Threat Prediction. Can we accurately distinguish threatening drone trajectories from benign operations using only initial flight segments? What is the minimum observation window required for reliable classification in operational settings? What features prove most predictive of hostile intent? Addressing these questions is critical because urban drone threats unfold in less than 5 minutes from launch to impact. Effective defense requires threat prediction early in the trajectory to enable meaningful response options.

Research Agenda 2: Conditional Synthesis of Threatening Trajectories. Given the scarcity of labeled threatening drone trajectory data, can we generate synthetic datasets that exhibit realistic spatial patterns, temporal dynamics, and threat-specific characteristics while capturing adversarial decision-making processes? How should trajectory synthesis be conditioned on geography, asset locations, and threat intent to ensure

generated data represents both benign and threatening behaviors appropriately? What validation methodologies establish that synthetic trajectories improve predictive model performance despite limited ground truth data?

Research Agenda 3: Legally Compliant Multi-Objective Decision-Making. How can we formalize international humanitarian law (126; 42), domestic aviation regulations, and rules of engagement into computational representations enabling automated reasoning in real-time defensive scenarios? When multiple defensive objectives conflict, how can autonomous systems efficiently identify Pareto-optimal sets of actions balancing competing considerations (95)? What computational frameworks provide both legal compliance guarantees and actionable decision support under operational time constraints?

Research Agenda 4: Constrained Reinforcement Learning for Defense. Can reinforcement learning agents learn effective multi-agent counter-drone policies through interaction with realistic simulation environments while maintaining strict compliance with legal constraints throughout the learning process? What architectural approaches enable the integration of symbolic legal reasoning with neural policy learning? How should reward structures be designed to encourage tactically sound defensive behaviors without incentivizing norm violations?

These 4 research agenda items form an interconnected framework where advances in one area enable progress in others. Early threat prediction provides a foundational capability upon which decision-making systems build. Legally compliant decision frameworks guide the design of learning objectives. Synthetic data generation improves both threat prediction accuracy and the diversity of scenarios encountered during policy training.

1.3. Research Approach and Contributions

As mentioned in the previous section, this dissertation presents an integrated framework for responsible urban drone defense addressing the research agenda through 4 interconnected contributions. We now elaborate on them.

1.3.1. DEWS: Drone Early Warning System

The first contribution introduces the Drone Early Warning System (DEWS), a framework for predicting whether drone trajectories are threatening based on initial flight segments (34). DEWS directly addresses Research Agenda 1 by demonstrating that accurate threat classification is achievable within 30 seconds of flight commencement, providing defenders with actionable intelligence while threats remain distant from potential targets.

DEWS' predictive approach integrates multiple feature categories capturing trajectory characteristics, drone capabilities, geography, and operational context. Basic trajectory features describe flight duration, distance traveled, and altitude profiles. Drone capability features characterize payload capacity, maximum velocity, and endurance. No-fly zone features quantify regulatory compliance. Critically, DEWS incorporates asset-based features assessing the value of ground locations overflown, recognizing that threat severity depends on what the drone threatens.

The DEWS architecture employs an ensemble of 11 classifiers spanning logistic regression, support vector machines, random forests, gradient boosting, and neural networks. Each classifier undergoes feature selection to identify the most effective prediction features. Late fusion combines classifier predictions, leveraging the complementary strengths

of different model families. The complete technical implementation and experimental methodology are detailed in Chapter 2.

Extensive experimental validation using 8 months of real drone trajectory data collected over The Hague by Dutch Police demonstrates DEWS's effectiveness. The dataset comprises 349 trajectories spanning recreational flights, commercial operations, and potentially threatening scenarios. Each trajectory received a threat score on a scale from 1 to 10 based on assessments by police officers and municipal security officials, with Cohen's kappa of 0.772 indicating substantial inter-annotator agreement. Using only the first 30 seconds of trajectory data, DEWS achieves F1-scores exceeding 0.85 for high threat classification.

Ablation studies reveal that asset-based features, specifically the maximum value of assets overflown, constitute the single most predictive feature category. This finding highlights the importance of geographic context in threat assessment and motivates the geographic conditioning approach employed in synthetic trajectory generation.

Systematic evaluation across observation windows from 5 seconds to 6 minutes characterizes the fundamental tradeoff between earliness and accuracy. Prediction performance improves rapidly during the first 30 seconds, plateaus between 30 seconds and 2 minutes, and exhibits minimal improvement beyond 3 minutes. This characterization provides actionable guidance for operational deployment, suggesting defensive systems should commence threat assessment at approximately 30 seconds post-launch.

1.3.2. STATE: Synthetic Trajectory Generation

The second major contribution addresses data scarcity through STATE (Safe and Threatening Adversarial Trajectory Engine), a conditional generative modeling framework for synthesizing realistic drone trajectories tailored to specific urban environments and threat contexts. STATE addresses Research Agenda 2, recognizing that effective machine learning in drone defense is fundamentally constrained by limited access to training data, particularly for threatening operations.

STATE's generative architecture conditions trajectory synthesis on 3 categories of information: geographic context, threat intent, and stochastic variation. The geographic context tensor encodes satellite imagery, road networks, building footprints, population density, no-fly zone boundaries, and asset value maps for the urban region of interest. This rich spatial representation enables STATE to generate trajectories that respect topographic constraints, follow plausible navigation patterns, and exhibit threat-appropriate relationships to valuable assets. The threat intent label explicitly conditions the generative process to produce trajectories matching the specified category. Latent noise variables seed diversity, ensuring multiple synthesis runs produce distinct trajectories.

STATE's novel architecture employs a variational autoencoder (VAE) architecture (58) modified to incorporate conditional information at multiple stages. The encoder network maps real trajectory exemplars into a learned latent space while conditioning on geographic and intent information. The decoder network generates trajectories from latent samples, conditioned on geography and intent. A novel critic module derived from DEWS' threat classification components provides additional supervision during training, encouraging the generator to produce trajectories that DEWS would correctly classify.

This threat-alignment loss ensures synthetic threatening trajectories exhibit characteristic patterns: proximity to sensitive assets, violations of airspace restrictions, and unusual trajectory profiles. Chapter 3 provides comprehensive technical details of the architecture and training procedures.

We validate the proposed method on a combination of quantitative distribution matching metrics, qualitative expert assessment, and downstream task performance evaluation. Distribution matching evaluates whether synthetic trajectories' statistical properties align with real data across both threatening and benign categories. Expert assessment by security personnel manually evaluates the qualitative realism and threat characteristics of individual synthetic trajectories. Downstream task evaluation demonstrates that threat prediction models trained on augmented datasets combining real and synthetic trajectories outperform models trained only on real data, with improvements being most pronounced for data-starved urban regions.

In summary, STATE extends the frontier of synthetic data generation for security applications by demonstrating that conditional synthesis of adversarial behaviors is feasible when appropriately constrained by domain structure. The ability to generate threat-conditioned trajectories for arbitrary urban regions enables defensive system development and testing in locations where real threat data does not exist.

1.3.3. POSS: Pareto-Optimal Status Sets for Legally Compliant Decision-Making

The third contribution addresses Research Agenda 3 through the Pareto-Optimal Status Sets (POSS) framework, which formalizes decision-making for autonomous defensive

systems in the presence of legal and ethical compliance requirements (36). POSS recognizes that effective drone defense requires not merely the technical capability to intercept threats but also systematic reasoning about the legal permissibility, ethical justification, and operational appropriateness of defensive actions.

POSS operationalizes legal and regulatory reasoning through 3 formal components: deontic logic for representing normative concepts (50; 79; 53), multi-objective utility functions for quantifying competing objectives, and Pareto optimization for identifying non-dominated action sets (95). The deontic logic component employs modal operators (50; 79; 53), obligation (**O**), permission (**P**), and prohibition (**F**), to express legal constraints in formal terms amenable to automated reasoning (105).

The multi-objective utility component recognizes that defenders simultaneously pursue multiple goals: neutralizing threats, minimizing collateral damage to civilians and infrastructure, preserving defensive capabilities, and maintaining proportionality between threat severity and defensive response. POSS represents these objectives through separate utility functions, explicitly maintaining the distinction among objectives and enabling decision-makers to analyze tradeoffs.

The Pareto optimization component identifies the set of actions that are non-dominated with respect to multiple objectives. In multi-objective optimization, a solution is Pareto-dominated if there exists another solution that improves at least one objective without degrading any other (95). The Pareto-Optimal Status Set represents the frontier of achievable outcomes where improving one objective necessarily requires sacrificing another. Importantly, POSS computes this set subject to legal constraints expressed in deontic logic, pruning candidate actions to retain only those complying with all applicable prohibitions

and obligations before evaluating their placement on the Pareto frontier. This two-stage process ensures the system never presents legally impermissible actions as viable options. The formal foundations and algorithmic implementations are presented in Chapter 4.

Experimental validation of POSS employs autonomous vehicle scenarios as a testbed. The validation uses highway driving scenarios where autonomous vehicles must navigate multi-lane traffic while optimizing multiple objectives including lane shift penalties, exit miss penalties, and speed change penalties, all while complying with traffic regulations encoded in deontic logic. Comparative evaluation demonstrates that POSS algorithms achieve substantial performance gains while maintaining legal compliance guarantees. While this validation demonstrates POSS's effectiveness in the autonomous vehicle domain, the framework's principles of integrating deontic constraints with multi-objective optimization apply directly to drone defense scenarios.

1.3.4. GUARDIAN: Learning-Based Defense with Legal Compliance

The fourth contribution synthesizes the preceding components into GUARDIAN (Governance-Unified Aerial Reinforcement-Defense In Accordance with Norms), a reinforcement learning-based defensive system designed to learn effective counter-drone policies while maintaining legal and regulatory compliance. GUARDIAN directly engages Research Agenda 4 by demonstrating that adaptive learning and the satisfaction of legal requirements are compatible objectives.

GUARDIAN's architecture integrates 3 major components: a Markov Decision Process (MDP) formulation (119) representing the sequential decision-making problem faced by defenders, a multi-agent reinforcement learning algorithm discovering effective policies

through simulated experience, and a POSS-based action pruning mechanism restricting policy search to legally compliant behaviors. The MDP formulation captures state space encompassing positions and status of friendly and adversarial assets, observations from sensor networks, and environmental factors. The action space includes movement commands for defensive drones, sensor tasking decisions, and engagement authorization. The reward function incentivizes threat neutralization while penalizing collateral damage, resource expenditure, and mission failures.

The multi-agent reinforcement learning component employs deep Q-networks (103) extended to multi-agent settings, enabling coordination among heterogeneous defensive assets with different capabilities and information access. Centralized training with decentralized execution (107) allows agents to share experiences during learning while executing independently during deployment.

The integration with POSS represents GUARDIAN's most distinctive innovation. At each decision point during both training and execution, POSS evaluates candidate actions available to defensive agents, pruning to retain only those satisfying legal and regulatory constraints given the current state. The reinforcement learning agent selects from this constraint-satisfying action set, ensuring learned policies never explore legally impermissible actions. This hard constraint enforcement contrasts with reward shaping approaches where constraint violations incur penalties but remain possible. GUARDIAN's approach provides formal guarantees that deployed policies satisfy legal constraints by construction.

Validation employs extensive experiments within a purpose-built 2D grid-based testbed across diverse configurations. The testbed simulates urban environments as grids of varying sizes, with experiments systematically varying the number of Blue team drones, the

ratio of Blue to Red drones, communication uncertainty with headquarters, and the complexity of deontic constraints. Results demonstrate that GUARDIAN successfully learns effective defensive policies while maintaining legal compliance. Comparison against unconstrained reinforcement learning baselines reveals that integrating legal constraints early in the learning process produces policies with more stable training dynamics and, in certain scenarios particularly at larger problem scales, comparable or superior performance to unconstrained approaches. This counterintuitive finding suggests hard constraints can facilitate learning by focusing exploration on viable policy regions. The complete experimental framework and results are detailed in Chapter 5.

GUARDIAN's contribution advances safe reinforcement learning by demonstrating that meaningful ethical and legal constraints can be formally integrated into the learning process for complex multi-agent decision problems.

1.4. Dissertation Structure and Chapter Organization

This dissertation is organized into six chapters presenting an integrated framework for responsible urban drone defense. Following this introduction, Chapter 2 presents DEWS for early threat prediction from minimal trajectory observations, while Chapter 3 introduces STATE for conditional synthesis of realistic threat-conditioned trajectories to address data scarcity. Chapter 4 develops the POSS framework for legally compliant multi-objective decision-making using deontic logic and Pareto optimization. Chapter 5 extends the contribution into GUARDIAN, demonstrating that reinforcement learning agents can learn effective multi-agent defensive policies while maintaining strict legal compliance through POSS-based constraint pruning.

The concluding chapter synthesizes contributions, discusses limitations, and articulates future research directions. During this research, we also developed DUCK (Drone Urban Cyberdefense) (35), a photo-realistic three-dimensional simulation testbed built on the Unreal Engine, providing supporting infrastructure for future system-level integration and validation in operationally realistic environments. While DUCK does not constitute a primary contribution, it demonstrates the feasibility of implementing the dissertation's components in realistic simulation settings and represents an important platform for future work integrating DEWS threat predictions with GUARDIAN-learned policies operating in three-dimensional urban environments.

1.5. Expected Contributions and Impact

This dissertation makes several contributions to the fields of security, artificial intelligence, and autonomous systems.

Technical Contributions: The work advances threat prediction through DEWS's demonstration that accurate classification is achievable from minimal trajectory observations using real-world law enforcement data. STATE extends generative modeling techniques to trajectory generation for security applications, introducing geographic and threat conditioning that addresses the critical data scarcity problem. POSS provides a principled framework for integrating legal constraints with multi-objective optimization, offering both theoretical rigor and practical algorithms. GUARDIAN demonstrates the architectural feasibility of reinforcement learning with hard constraint enforcement for multi-agent coordination problems.

Empirical Contributions: Extensive experimental validation using real trajectory data from European law enforcement, expert assessments by security personnel, and systematic component-level evaluation provides empirical evidence for the feasibility of responsible urban drone defense. DEWS's characterization of tradeoffs between earliness and accuracy offers actionable insights for operational deployment. STATE's synthetic data generation demonstrates measurable improvements in threat prediction when used for training augmentation. POSS's identification of Pareto-optimal action sets illustrates the practical value of multi-objective optimization under constraints. GUARDIAN's validation establishes the architectural soundness of constrained reinforcement learning.

The framework presented represents meaningful progress toward responsible urban drone defense systems designed to be simultaneously effective at neutralizing threats, compliant with legal and regulatory constraints, adaptive to evolving adversarial tactics, transparent in their decision-making processes, and subject to meaningful human oversight. This research demonstrates that these objectives are compatible and that progress toward their realization is possible through rigorous interdisciplinary inquiry bridging computer science, security studies, legal scholarship, and ethics.

CHAPTER 2

A Drone Early Warning System (DEWS) for Predicting Threatening Trajectories

Over the last few years, there has been increasing use of drones by terror groups and in armed conflict. Several technologies have been developed to detect drone flights. However, much less work has been done on the Drone Threat Prediction Problem (DTPP): predicting which drone trajectories are threatening and which ones are not. We propose DEWS (Drone Early Warning System), a framework to solve this problem. Solving DTPP early is key. Once a drone starts on its trajectory, we show that DEWS can make accurate predictions within 30 seconds of the flight with an F1-score of over 80% on data about a major European city. We study the tradeoff between earliness of predictions and accuracy. We identify the key features that ensure good predictions.

2.1. Introduction

Terror groups such as ISIS (3), the PKK (108), Lashkar-e-Taiba[1], and others are increasingly using drones in various operations. Drones are also becoming a preferred instrument of nation state warfare as evidenced by the war in Ukraine. There is now deep concern that cities will be targeted by drone attacks (13).

[1] https://www.indiatoday.in/india/story/drone-attack-initial-probe-lashkar-role-jammu-and-kashmir-police-chief-1820679-2021-06-29

However, the skies over a city are traversed by numerous drones. Realtors use drones to get aerial shots of properties for sale(125), insurance companies use drones to look for undeclared pools and property damage (124), sports arenas use drones to capture crowd pictures and game plays (132), and more. A major problem for police and security organizations around the world is to distinguish the few drones that pose a threat from the many that are benign. And we need to do this as early as possible. As stated by defense experts at the Modern War Institute at West Point[2], "The earlier you detect a threat (drone, rocket, missile, or artillery), the sooner you can alert the force to seek shelter while the air defense operators work to employ their systems to defeat the threat".

This is the problem that we address in this chapter: developing a machine learning model that takes an *initial part* (e.g. the first 5, 10, 20, 30 seconds, ...) of a drone trajectory as input and predicts if it is threatening or not. The smaller the "initial" part, the earlier we can bring a potentially threatening trajectory to the attention of security agencies. But a small initial part might be too small to make a good prediction.

Though there has been a great deal of work on predicting trajectories of moving objects (e.g. mobile phones (65), drones (101)), there has been relatively little work on quantifying the *threat* posed to a city or geographic area by a drone. To quantify this threat, we must not only understand the drone's trajectory, but also the drone's capabilities (e.g. payload, battery life, max speed) and the value of the assets on the ground that the drone is flying over.

[2] https://mwi.westpoint.edu/understanding-the-counterdrone-fight-insights-from-combat-in-iraq-and-syria/

Our DEWS Drone Early Warning System predicts whether a drone trajectory is threatening or not. DEWS tries to understand how long we need to observe a drone flight in order to predict whether the drone poses a threat or not.

DEWS is novel in several respects. (i) As far as we know, DEWS is the first framework to predict the *threat* a drone flight poses to a city. (ii) It is the first framework to understand the tradeoff between the time for which a drone trajectory is observed (the "observation window") and threat prediction accuracy. (iii) In addition to the trajectory, DEWS looks at features about the drone's capabilities, violations of no fly zones, assets on the ground, and more. (iv) DEWS identifies the key features linked to accurate predictions. We find that the values of assets on the ground that a trajectory flies over constitute the single most important feature in assessing the threat of the trajectory. (v) DEWS can make predictions with an F1 score exceeding 0.8 in 30 seconds in operational use (after training), suggesting that it can be used for real-time predictions. (vi) DEWS has been tested by Dutch police, municipal, and security officials on 8 months of real trajectories over The Hague and the results show an F1-score over 0.85.

This chapter is organized as follows. The "Related Work" Section discusses related work. Next, Section "DTPP: Drone Threat Prediction Problem" formalizes the problem studied. Our "DEWS Architecture" Section provides a detailed description of our architecture, including its features and training process. Section "Experiments" presents the predictive performance of 11 ML models and a late fusion classifier) as the observation (i.e. training) window increases. After this, a "Limitations and Future Work" section describe limitations of the framework.

2.2. Related Work

Predicting the future location of a moving object has been explored in various domains (52; 97). Vision-based object tracking methods (25) predict the future location of moving objects. This work has been used in self-driving cars (97) to create plans based on predicted future locations of humans and nearby moving objects. Other research uses historical GPS data to predict mobility of devices (65).

Numerous papers predict vehicle trajectories by learning models from historical driving data (14). Temporal models such as LSTMs with attention networks (68; 83; 104) have been proposed for trajectory prediction. Recent advances incorporate trajectories of nearby vehicles to reduce accidents (64). Drone trajectory prediction has been widely studied across various applications, including autonomous aerial cinematography (17), delivery (101), and search and rescue (1).

There is also work on predicting a mobile phone's next location based on historical movement data (92; 91). These approaches include sequential pattern learning techniques to predict a phone's future location and/or human movements.

DEWS differs from past efforts in two respects. First, it predicts if a drone trajectory is *threatening* or not, which past works don't do. Second, DEWS is the first to study how early in a trajectory we can make a good prediction. This is particularly important because timeliness is key in mitigating drone threats. The identification of a threat is a crucial input for the subsequent command and control process resulting in some kind of intervention. DEWS not only obtains features from the drone trajectory, but also from assets on the ground and the drone's capabilities. Past work doesn't consider assets on the ground.

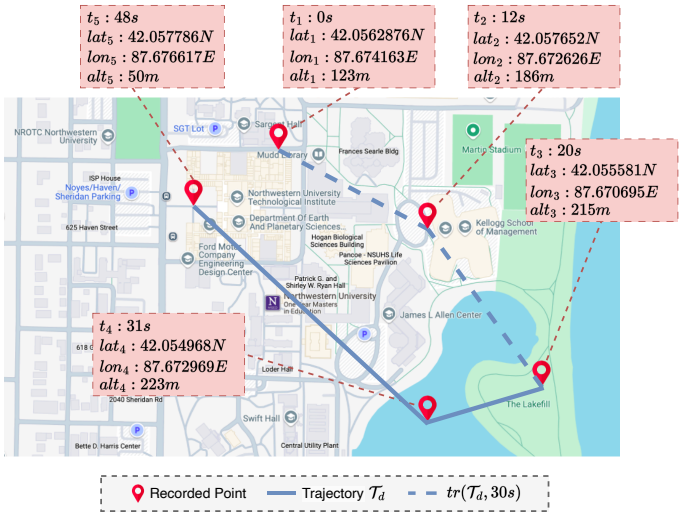


Figure 2.1. Sample drone trajectory with its 30-second restriction. The trajectory data is from a real drone, but the city was altered for security reasons.

2.3. DTPP: Drone Threat Prediction Problem

Suppose C is a city to be protected. We obtain a map of C containing locations of important national buildings, security installations (e.g., police stations, military bases), government buildings, hospitals, tourist attractions, entertainment venues, homes, parks, roads, bridges, utilities, etc.[3] Once the city C is selected, we define an asset valuation map $Val(C)$, which assigns a value to every point within the city. High $Val(C)$ values corresponds to important locations.

Consider a drone d flying over C. Its *trajectory* τ_d is a finite sequence $(\ell_1^d, t_1), \ldots, (\ell_n^d, t_n)$ where each $\ell_i^d = (lat_i, long_i, alt_i)$ is d's location at time t_i in terms of latitude, longitude and altitude, respectively. The *temporal restriction* of a trajectory τ_d to time j, denoted $tr(\tau_d, j)$, is the set $\{(\ell_i^d, t_i) \mid (\ell_i^d, t_i) \in t \wedge t_i \leq j\}$. We use $\mathcal{T}$ to denote a given set of trajectories and we use $tr(\mathcal{T}, j) = \{tr(\tau, j) \,|\, t \in \mathcal{T}\}$ to be the restriction of the trajectories

[3]We used OpenStreetMap https://www.openstreetmap.org.

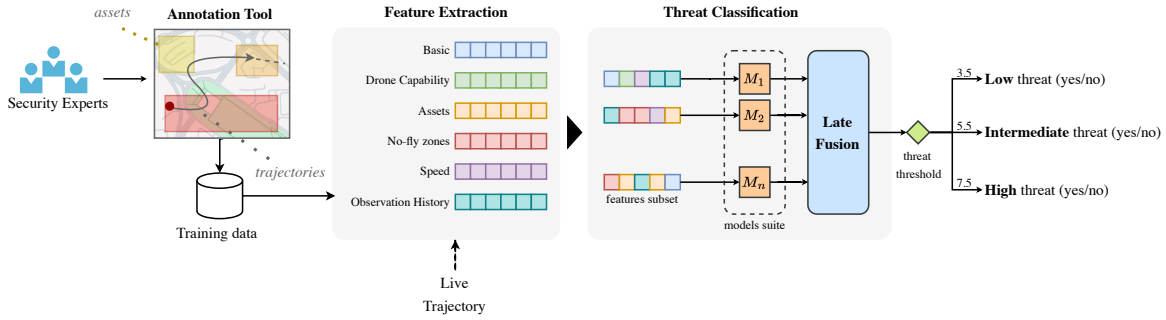


Figure 2.2. DEWS Architecture. Data set preparation involves annotating *asset values* and *drone trajectories* by *police*. Subsequently, DEWS extracts *features* and trains 11 classifiers $M_1, \cdots, M_{11}$ to yield 11 predictions which are integrated using *late fusion* to predict the final *threat level*. During operational use (after training), an initial part of a *live trajectory* is processed to extract features, and the combination of single predictors and late fusion produces the final threat score.

in $\mathcal{T}$ to the first j timepoints. Figure 2.1 shows a drone's trajectory τ_d and its restriction $tr(\tau_d, 30)$ to 30 seconds.[4]. As an example, we may wish to predict the level of threat posed by $tr(\tau_d, 30)$ after the 30 seconds of the flight. The threat score is given by $y(\tau_d) \in [1, 10]$. The higher the threat score, the more threatening the drone's trajectory.

The Drone Threat Prediction Problem (DTPP[lev]) is to learn a function f_{lev} : $(d, tr(\tau_d, j)) \rightarrow \{0, 1\}$, such that $f(d, tr(\tau_d, j)) = 1$ if the threat posed by $\tau_j \geq lev$, where $lev \in [1, 10]$.

DTPP can work after any observation window $j > 0$ after the drone flight begins. This is critical for security. The earlier predictions are made about the threat level of trajectories, the earlier security officials can prioritize their responses.[2]. Earliness of prediction must be balanced against accuracy of prediction. Understanding this balance is a major goal of this chapter.

[4]Drone locations may be acquired at irregular intervals.

Table 2.1. DEWS Dataset Statistics

Statistic	Threat Score [1, 3]	[4, 7]	[8, 10]
Number of Drones		18	
Number of Trajectories	213	94	42
Avg. Duration (s)	265	298	286
Avg. Distance (m)	435.1	988.2	752.1
Avg. Altitude (m)	62.69	115.9	100.7
Avg. Speed (km/h)	7.088	14.58	10.41

2.4. DEWS Architecture

Figure 2.2 shows the DEWS architecture. DEWS uses a dataset of drone trajectories annotated by Dutch police and municipality — Table 2.1 presents a brief overivew.

The *Feature Extraction* module extracts key features that characterize a drone trajectory. The *Threat Classification* module combines the predictions of 11 classifiers to provide a final classification.

2.4.1. Trajectory Training Data

We collected a dataset of 349 drone trajectories to train DEWS. These trajectories represent all known recorded drone flights over The Hague captured by Dutch police and municipality over a period of eight months. The threat of each trajectory was assessed on a 1-10 scale by at least one police official. **50 trajectories were annotated independently by 2 or more police and municipal officers. To assess agreement amongst the officials, we computed the inter-annotation weighted Cohen's kappa coefficient of 0.772, indicating substantial agreement amongst annotators.**.

Police officials then categorized trajectories as *low* threat (score < 4), *medium* threat (score $\in [4, 8)$), and *high* (score ≥ 8) threat.

2.4.2. Feature Extraction

This module extracts 110 features for each trajectory. Appendix A describes all the features.

Basic features offer an initial summary of each trajectory. They include the number of observations, duration of the flight, distance traveled, and communication channel used (e.g. radio-frequency, Wi-Fi).

Capability features include physical attributes (e.g., weight, dimensions) and performance specifications (e.g., maximum payload, battery capacity). These features are critical for assessing the drone's operational limits and the potential threat it may pose.

Altitude features (e.g., the mean altitude above takeoff) and *speed features* (e.g., minimum/maximum speeds) provide insight into the dynamics of each trajectory. They are essential for detecting suspicious activities and ensuring regulatory compliance as drones may have altitude or speed restrictions.

No-fly Zone features capture the behavior of trajectories in terms of their respect for the law. We used no-fly zone data[5], and defined six features to quantify the proximity of the trajectory to a no-fly zone, e.g., whether the drone entered a no-fly zone, the percentage of time the trajectory was within a no-fly zone.

Asset-based features (e.g. dams, utilities, government buildings, defense sites) also need to be considered when assessing the threat of a trajectory. We defined features

[5]https://www.godrone.nl

about the proximity of the trajectory to these assets (e.g. the maximum/mean asset values overflown). Asset values were provided by Dutch police.

Observation history features capture the similarity between the current trajectory and historical trajectories. *Self-similarity features* refer to the similarity between the current trajectory and past trajectories of the same drone, which can help detect recurring flight patterns or behaviors that may indicate potentially benign operations. *Cross-similarity features* capture the similarity between the current trajectory and past trajectories of other drones. This may be useful for identifying anomalous behavior by comparing it to known suspicious or dangerous flight patterns exhibited by other devices. Cosine similarity is used in both.

2.4.3. Threat Classification

The Threat Classification module predicts the threat level (low, medium, high) of a trajectory based on its extracted features. To accomplish this, we trained a suite of 11 well-known machine learning classifiers, encompassing both traditional and neural network models.[6]

For each classifier, we did hyper-parameter optimization and applied feature selection to identify the most relevant subset of features. The feature selection process consists of three main steps: (1) removing constant columns, (2) retaining only one feature from pairs of features with a Pearson correlation greater than 0.95, and (3) selecting the top-k features based on their Mutual Information (MI) scores, where k is a user-defined

[6]The classifiers used in DEWS are: Logistic Regression, k-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Trees, Random Forest, Gradient Boosting, Naive Bayes, AdaBoost, Extra Trees, a Multi-layer Perceptron (MLP), and a wide-and-deep neural network.

parameter. This approach allowed us to develop specialized models that leverage distinct subsets of features for the same trajectory, thereby enhancing model diversity within the suite.

After individually training each model M_i, we used late fusion to combine their predictions. The final threat score for a trajectory t is computed as a weighted sum of the probability estimates produced by each model: $y(t) = \sum_{i=1}^{11} M_i(t) \cdot w_i$, where $M_i(t)$ represents the probability prediction of model M_i that trajectory t is threatening, and w_i denotes the weight assigned to model M_i. The weights w_i were optimized through grid search to identify the combination of weights that maximized overall classification performance. This fusion process enables DEWS to integrate the strengths of multiple models, ensuring robust and accurate threat classification.

2.5. Experiments

All experiments were conducted on a computational platform having a 10th Gen Intel i9-10980XE processor, 256 GB of RAM, and an NVIDIA RTX A6000. The codebase involved approximately 2000 lines of code in Python 3.10. All classification models were implemented using the Scikit-learn library, expect for the wide and deep classifier for which we used the Tensorflow 2 library.

2.5.1. Data Collection

Data about 349 drone trajectories over a Dutch city was systematically collected by the Dutch police using the Senhive[7] commercial drone tracking system. This system tracks drones by monitoring their communication frequencies with drone operators, allowing

[7]https://senhive.com/sen-id-1

for the detection and recording of their trajectories within a radius of 25 km. The Senhive system provides the device model name (e.g., DJI Mini 3 Pro) for drones detected within its operational range. Based on this information, we derived the capability features by referencing the manufacturer-provided specifications for each identified model. An anonymized version of this dataset was provided by the Dutch Police to the academic part of our team, with sensitive information such as device IDs replaced with anonymized IDs. Summary statistics for the dataset are provided in Table 2.1.

We developed our own GUIs for annotating asset values and threat scores associated with the drone trajectories. Each trajectory was individually assessed and annotated based on its specific characteristics and potential threat level by 2 police officials and 2 security officials from the Hague Municipality, all with deep experience in drone threat assessment.

2.5.2. Experimental Protocol

In our experiments, we address the DTPP problem at three distinct levels: 3, 5 and 7. This corresponds to the scenarios detailed as follows:

(i) *Low-Threat Prediction* (LTP): trajectories with a threat score in the $[1, 4)$ range (i.e. greater than or equal to 3 and strictly less than 4) are considered low threats. The LTP problems predicts no-threat (score less than 3) and low threat (score greater than 3). As you can see from Table 2.1, 213 of 349 trajectories (61.03%) in our dataset were considered to be low threat which was consistent with what our security experts had seen in their real-world assessments.

(ii) *Medium-Threat Prediction* (MTP): trajectories with a threat score in the $[4, 8)$ range are considered medium threat trajectories. So MTP distinguishes between medium threats (score of 4 or more) and other trajectories. Table 2.1 shows that 94 of 349 trajectories (26.93%) in our dataset posed a medium threat.

(iii) *High-Threat Prediction* (HTP): trajectories with a threat score greater than or equal to 8 are classified as threatening. Finally, Table 2.1 shows that the other 42 trajectories (12.03%) in our dataset posed a high threat.

By applying the learned predictive models for a given trajectory, we can uniquely classify a trajectory into one of the four threat levels (no threat, low, medium, high threat).

These classification tasks are increasingly difficult due to the skewed distribution of threat labels, with the HTP setting containing significantly fewer threatening trajectories compared to MTP and LTP.

We conducted three experiments:

- *Early Threat Prediction Evaluation*: We assess DEWS's capability for early threat prediction by varying the observation window for each trajectory. Specifically, we analyze each trajectory t using the first i seconds of a flight, where $i \in \{1, 5, 10, 20, 30, 60, 180, 360, 720\}$. This assesses how early accurate predictions about the potential threat can be made.
- *Ablation Study*: To determine the relative importance of different feature types, we systematically remove each feature type from the model and retrain the DEWS[lf] late fusion predictor. Performance is then evaluated based on recall,

precision, and F1-scores to identify which features contribute most significantly to predictive accuracy.

- *Feature Relevance Analysis*: Assuming that features selected for classification are the most relevant for solving the task, we analyze the features chosen by each classifier during the feature selection process. For each observation window, we count how often each attribute is selected for classification across all classifiers in the model suite. These counts are then normalized to compute the relative frequency of each feature category. Specifically, let w denote an observation window, $\mathcal{A} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n\}$ represent the set of features, and $M_1, M_2, \ldots, M_{11}$ be the classifiers in the model suite. We define:
  - $N_{ij}^{(w)}$ as the count of how often the feature $\mathcal{F}_i$ is selected for classifier M_j during M_j's features optimisation process for classification, within window w;
  - $N_i^{(w)}$ as the total count of how often the feature $\mathcal{F}_i$ is selected across all classifiers, i.e.
$$N_i^{(w)} = \sum_{j=1}^{11} N_{ij}^{(w)}.$$

  To compute the relative frequency $f_i^{(w)}$ of the feature $\mathcal{F}_i$ for the observation window w, we normalize $N_i^{(w)}$ by the total counts for all attributes:
$$f_i^{(w)} = \frac{N_i^{(w)}}{\sum_{k=1}^{n} N_k^{(w)}}.$$

- *Runtime*: We measure DEWS's runtime for feature extraction and classification with late fusion in operational use (after training).

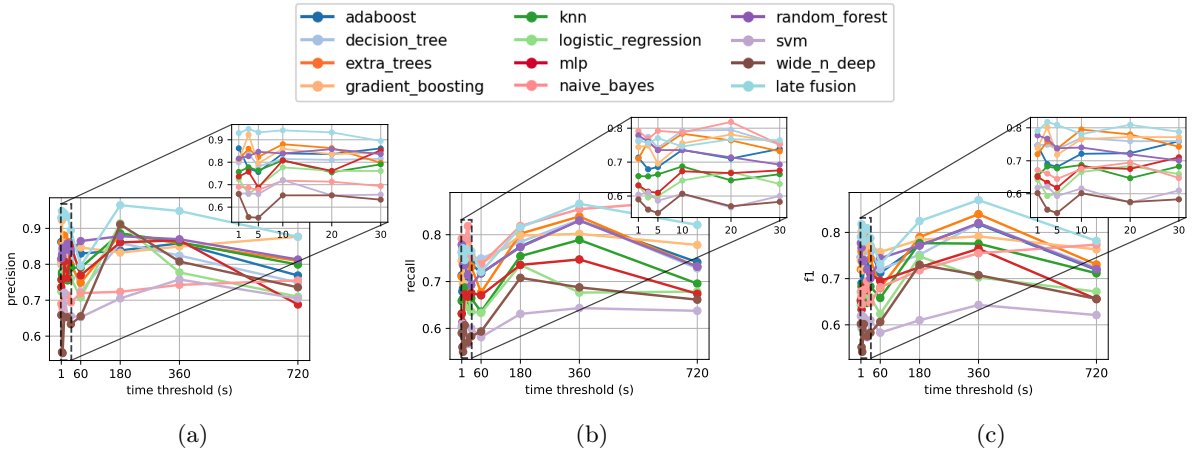


Figure 2.3. High-Threat Prediction (HTP) settings: Precision (a), Recall (b), and F1-score (c) metrics are shown as functions of varying temporal restrictions on the trajectories. The top row provides a zoomed-in view of the results for shorter time windows (less than 30 seconds), while the bottom row displays the complete range of observation windows.

All experiments were conducted using time series cross-validation, i.e. we learned a model from an early set of trajectories and then used them to predict on later sets of trajectories.

2.5.3. Results

Early Threat Prediction Evaluation. Figure 2.3 illustrates DEWS's performance under the HTP setting. This setting poses the biggest challenge in our work because of class imbalance (12.03% highly threatening, 87.97% not highly threatening) which is well-known to be difficult to handle.

Figures 2.3a, 2.3b, and 2.3c depict precision, recall, and F1-score, respectively, for all 11 classifiers as well as the DEWS late fusion classifier, DEWS[lf]. These metrics are analyzed by varying the observation window. Performance comparisons under MTP and LTP settings are reported in the Appendix A.

Finding 1: Late Fusion is the Best Predictor. Late fusion consistently outperforms the 11 classifiers across all observation windows, achieving the highest results in terms of precision, recall, and F1-score. As shown in Figure 2.3, within an observation window of 30 seconds, DEWS[lf] stabilizes at an F1-score of approximately 80%, with precision exceeding 90% and recall around 75%. As the observation window increases, performance shows an upward trend, with the most substantial improvement occurring between one minute and three minutes. The best performance is achieved at the six-minute threshold, where precision reaches 0.967 and recall 0.869.

Finding 2: Increasing the Observation Window may Not Improve Performance. Interestingly, increasing the observation window does not always lead to improved performance. For example, Figure 2.3 shows that the highest recall of 0.789 for shorter observation windows occurs with 5 seconds of observation, when precision is 0.934 (using our DEWS[lf] classifier). Both metrics show a slight decline when the window is extended up to 30 seconds. Moreover, beyond six minutes, performance deteriorates across all models and metrics.

Finding 3: Precision is always higher than recall. Figure 2.3 shows that the same time thresholds yield higher performance in terms of precision compared to recall. For instance, with a short observation window of 5 seconds, precision reaches 0.934, while recall is comparatively lower at 0.789. This trend is consistently observed across all observation windows. This is due to the imbalance of the data considered for the HTP problem which causes DEWS to be more conservative when predicting the the highly threatening (minority) class. This suggests that DEWS is very accurate at detecting highly threatening trajectories with a very low false positive rate.

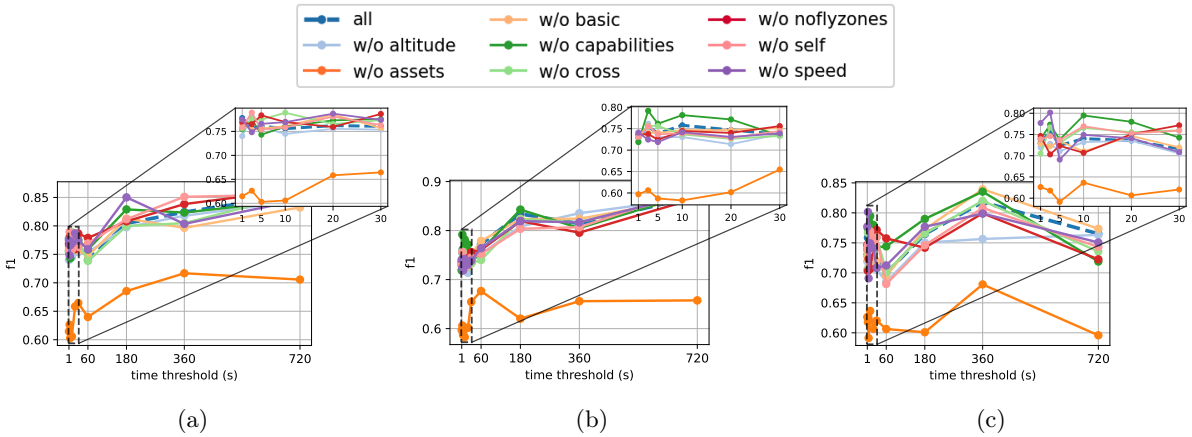


Figure 2.4. Ablation Study: F1-score under LTP (a), MTP (b) and HTP (c) settings when removing one feature category. The dashed line represents the scenario with all features.

This is extremely valuable for police for two critical reasons: First, it enhances trust in the system, as the low false positive rate minimizes the likelihood of unnecessary interventions. Second, in resource-constrained environments, human assessment of predicted high threat can be costly and inefficient. High precision ensures that humans don't get frustrated with false positives.

Ablation Study. Figures 2.4a, 2.4b and 2.4c show the F1-scores obtained when removing individual feature categories under the LTP, MTP, and HTP settings, respectively.

Finding 4: Asset-related features are the most critical for threat prediction. We see from Figure 2.4 that with a 5-second observation window, the F1-score with all features included is 0.723 in the HTP setting, but decreases to 0.586 when asset features are excluded, representing a 19% reduction in performance.

Equally surprising are the features that proved to be less important than we had expected. For example, we initially hypothesized that no-fly zone features would play a significant role in threat prediction, yet they had a relatively minor impact on the model's performance. Similarly, we expected the type of drone (e.g., fast drones with

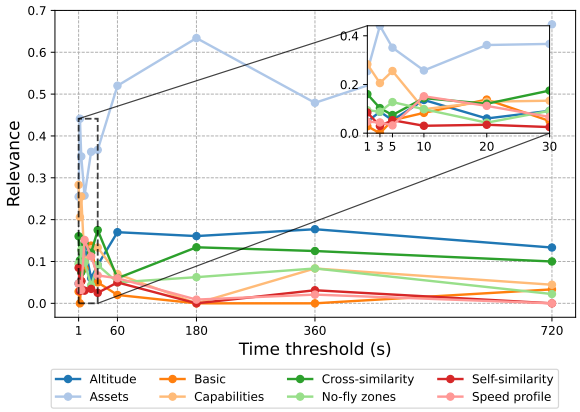


Figure 2.5. Feature Relevance Analysis (HTP problem): relative frequency of feature categories selected by classifiers across different temporal restriction windows.

large payloads) to be a key predictor, but their importance for prediction was small. Additionally, speed-related features, which we assumed would be important, turned out to have limited significance in our experiments.

Overall, these findings support our preliminary hypothesis that the geographical region, represented by asset-related features, is a key determinant in assessing the threat level of a trajectory, independent of the drone's intrinsic characteristics or the specific properties of the trajectory itself.

Feature Relevance Analysis. The results in Figure 2.5 (HTP problem) indicate that as the observation window increases, the importance of asset-related features becomes more pronounced. For instance, after a 180 second observation window, over 60% of the features used for classification belong to the asset category. Interestingly, within the first 1-5 seconds of observation, capability-related features show relatively high importance. The importance of these features decreases sharply with longer observation windows. This may be due to the limited information available in short observation windows, where the drone's capabilities alone serve as a strong indicator of potential threat.

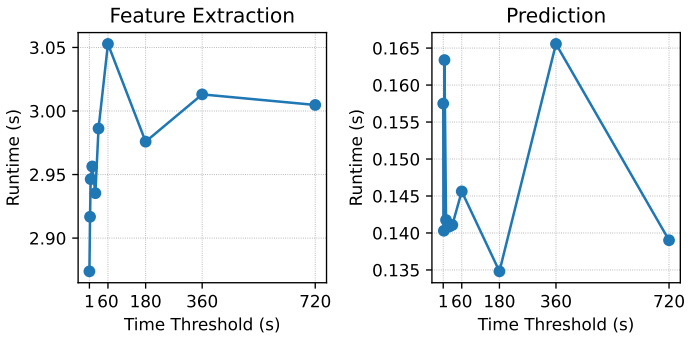


Figure 2.6. Runtime Analysis (HTP problem): Time (in seconds) for feature extraction (left) and prediction using late fusion (right).

Runtime. Figure 2.6 shows the mean DEWS runtime for feature extraction and prediction (with late fusion) under the HTP setting in operational use. The feature extraction time shows a slight increase with a larger observation window, reflecting the additional computational load due to the increased number of trajectory points. In contrast, the prediction time is not affected by the length of the trajectory. With an overall classification time of approximately 3 seconds, the DEWS system demonstrates its potential for real-time predictions, enabling trajectory classification after just 3-5 seconds of observation. This highlights the system's suitability for applications requiring prompt decision-making.

2.6. Limitations and Future Work

Like all studies, our study can be improved in many ways. First, we note that we looked at all trajectories over a city that Dutch police tracked over an 8-month period. Drone tracking technology (e.g. radar or acoustic sensors) that differs from the methods used by Senhive (which monitor drone communications frequences - but further details are proprietary to the company and we don't have access to it) may enable tracking flights that Senhive doesn't track. DEWS' predictive accuracy and operational effectiveness when

a different tracking method is used remains to be studied. That said, our results show that the single biggest factor in assessing drone threat is the assets in the city which doesn't change based on the drone. Second, our dataset includes trajectories from a single city in the Netherlands, posing challenges to generalization across diverse operational scenarios (e.g., warzones). For instance, detecting threatening vs. non-threatening drones in a battlespace (e.g. Ukraine) may differ. Yet, DEWS has been tested with three different distributions (LTP, MTP, HTP), so there is hope that the same principles apply - however, we do not have labeled information about drone trajectories in warzones. Third, once adversaries know about DEWS, they may take evasive actions to prevent their intentions being predicted. This is partly mitigated by our finding that asset value is the important feature in assessing threat - and adversaries cannot manipulate that. But the development of ML models that are more robust to an adversary's evasion attempts need to be studied as a next step. Fourth, two or more trajectories that individually seem non-threatening might collude to pose a significantly higher threat. This needs to be studied. Finally, DEWS is designed for use by public authorities only. While misuse (e.g. privacy violations) are possible, other excellent efforts such as (123; 122) can be combined with DEWS to mitigate such risks, e.g., by utilizing features that do not include information capable of identifying drones or revealing their start/end coordinates.

2.7. Conclusion

To the best of our knowledge, this is the first work to explicitly study the problem of how threatening a drone flight is to a city or geographic region. We propose a repertoire of features for quantifying the threat of a drone flight, build out the first drone threat dataset

that was assessed by police and security officials and will be made publicly available (with some anonymization to ensure security), and build the first predictive models to assess the threat level posed by a trajectory. We are also the first to show that we can predict threat levels *early*, when a trajectory is just underway. With just 30 seconds of trajectory data, DEWS is able to make predictions of high threat levels with an F1-score over 0.8. And these predictions take only a few seconds to make. Predictive accuracy goes up till about 5-6 minutes of the trajectory is observed. This enables DEWS to continuously provide forecasts to security officials after 30 seconds of the flight is observed and they can decide on their response depending on their own judgement and knowledge of context. DEWS also allows predictions to be tailored to a specific context and threat assessment. In other words, given a specific threat assessment, particular assets (on the ground) or capabilities (of drones) may be valued differently — and DEWS will still work.

Our biggest new finding is that the key determinant of the danger posed by a trajectory is not the trajectory itself, but the values of the assets on the ground that a trajectory flies over.

CHAPTER 3

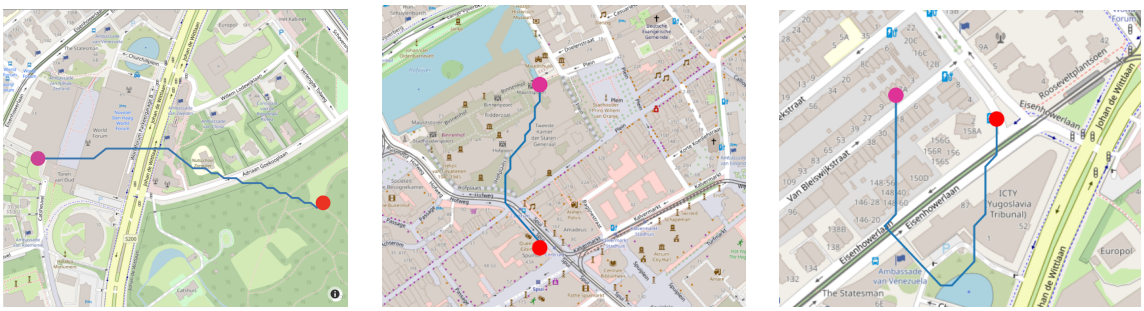
STATE: Safe and Threatening Adversarial Trajectory Engine

Chapter 2 demonstrated that accurate early threat prediction requires comprehensive training data spanning benign and threatening trajectories. However, operational drone monitoring infrastructure captures limited threatening flight examples, and privacy regulations may constrain data sharing across jurisdictions. To address the problem, we present *STATE*, a novel GAN-based framework to automatically generate a set of safe and threatening drone trajectories over a region. Using *STATE*, security officials can generate synthetic trajectories for regions over which no such trajectories were previously recorded, enabling them to better test planned defenses. Jointly with security officials from The Hague, we show that *STATE* beats five baselines, achieving up to 75.8% improvement in trajectory plausibility and 35.8% improvement in threat alignment, as evaluated by police experts.

3.1. Introduction

Cities are likely to have an increasingly complex airspace in coming years as the use of drones for product delivery, medical purposes, real-estate and insurance surveys, entertainment, urban sensing, and more, expand (101). This complex airspace will provide new threat vectors for terrorists and rogue nation states (96).

The recent DEWS (34) system looks at drone flights in real-time and predicts whether the flight will be safe or threatening after 30 seconds of observation of the flight. While this



(a) (b) (c)

Figure 3.1. Three examples of threatening trajectories generated with *STATE*. (a) Trajectory around *Park Sorghvliet* and surrounding districts, terminating within a no-fly zone. (b) Trajectory traversing a sensitive zone with multiple government buildings, also terminating in a no-fly zone. (c) Trajectory beginning in a residential neighborhood and performing perimeter surveillance around a sensitive institutional complex without entering no-fly zones.

is useful, DEWS was trained on 8 months of drone flight data from The Hague, Netherlands. However, many cities do not currently have drone tracking mechanisms in place. We leverage the DEWS dataset and collaborate with two law enforcement experts who assess drone threats daily to design *STATE* (*Safe and Threatening Adversarial Trajectory Encoder*), to automatically generate a set of safe and threatening drone trajectories over any geographical region. As long as someone with security knowledge of a geographical area can annotate the value of its assets on the ground (e.g., The Blue House in Seoul), *STATE* can automatically generate a set of trajectories, both safe and threatening, that are consistent with drone flight distributions that the DEWS (34) team observed over The Hague. Figure 3.1 shows three examples of threatening trajectories over The Hague generated with *STATE*. Using STATE, systems like DEWS can be trained to provide early warning threat assessments posed by drone trajectories to other regions (not just The Hague), enhancing public safety.

This chapter makes four contributions. *First*, we formally define the *Threat-Conditioned Trajectory Generation* task to synthesize drone trajectories conditioned on a specified

threat class within a given geographical area. *Second*, we develop the novel *STATE* framework for *Threat-Conditioned Trajectory Generation*. *STATE* uses a conditional Generative Adversarial Network (cGAN) architecture to generate spatially realistic trajectories that are aligned with a target threat class. A central innovation is the use of a pre-trained threat classifier, which serves as an auxiliary supervision signal to enforce threat consistency during generation. *Third*, we curate and release a new dataset of 200 synthetic drone trajectories over The Hague. Each trajectory is manually annotated by two law enforcement experts based in The Hague with corresponding threat level labels, providing a valuable resource for future research on security-aware trajectory modeling[1]. *Fourth*, our extensive experiments assess *STATE* against five baselines in terms of spatial plausibility, trajectory diversity, and semantic threat alignment of the generated trajectories. Our results show that *STATE* consistently outperforms all baselines, achieving relative improvements of up to 75.8% in spatial plausibility and 35.8% in threat classification consistency, as judged by police experts.

3.2. Related Work

Trajectory forecasting has looked at predicting future positions of Unmanned Aerial Vehicles (UAVs) or drones based on partial historical data (73). More recent efforts generate complete, synthetic, yet realistic flight paths from scratch (114).

Most works treat trajectory generation as a sequence modeling problem, where locations are generated sequentially, conditioned on a predefined start token/position. Markov models were used to capture local sequential dependencies in movement patterns (51).

[1]To promote reproducibility, we release the code repository and dataset: https://github.com/nsail-lab/STATE-Codebase

However, such models are unable to represent long-range temporal dependencies, which are essential to accurately reflect the dynamics of real-world UAV trajectories. To overcome these limitations, Long Short-Term Memory (LSTM) networks, have been widely adopted for autoregressive trajectory generation (113). These models capture long-term temporal dependencies and incorporate rich contextual features such as spatial coordinates, altitude, and time (135). Recently, Conditional Generative Adversarial Networks (cGANs) (134) have emerged as a promising framework to create trajectory data through adversarial training. In addition to preserving temporal coherence, these models support explicit conditioning on external variables. (62) generates trajectories aligned with predefined speed profiles, while (93) incorporates weather conditions into the generative process. (8) has also integrated device-specific requirements into the generative process, e.g., minimizing energy consumption, avoiding collisions.

An alternative line of research explores the generation of trajectories as images that preserve spatial structure. Trajectories are first synthesized as an image over a specific area and later converted into sequential data points. (9) generated trajectory images under a smoothness constraint and then used an LSTM to recover the ordered sequence of waypoints. (139) introduced a UAV 2D trajectory forecasting framework that leverages an attention-based aggregation module to capture short-term spatio-temporal dependencies among trajectory points.

To date, we are not aware of any UAV trajectory generation method that considers the potential security threats posed by UAV flight paths. To the best of our knowledge, the only work addressing security aspects in this context is presented in (34), which frames the problem as a classification task, distinguishing between safe and threatening *pre-existing*

trajectories. They do not, however, *generate either safe or threatening* trajectories which is the goal of this chapter.

3.3. Problem Formulation

A trajectory τ is defined as an ordered sequence of M_τ waypoints:

$$\tau = \left\{ w_j = \left(\text{lat}_j, \text{long}_j, h_j \right) \mid j = 1, 2, \ldots, M_\tau \right\},$$

where each waypoint w_j consists of a latitude lat_j, a longitude long_j, and an altitude h_j.

Let $\mathcal{D} = \{(\tau_i, \theta_i)\}_{i=1}^{N}$ denote a "training" dataset comprising N drone trajectories recorded within a given region $\mathcal{C}$. Each trajectory τ_i is annotated with a binary threat label $\theta_i \in \{0, 1\}$, denotes a safe (resp. *threatening*) trajectory when $\theta_i = 0$ (resp. $\theta_i = 1$).

A *planar projection*, $\hat{\tau} \in \{0, 1\}^{H \times W}$, of trajectory τ, over a 2-dimensional grid of size $H \times W$ ensures that each non-zero element in $\hat{\tau}$ indicates the presence of at least one waypoint of τ projected onto the corresponding cell.

The *Threat-Conditioned Trajectory Generation* problem seeks to learn a generative model $\mathcal{G}$ capable of synthesizing drone trajectories that are conditioned on a target geographical area for both $\mathcal{A} \subseteq \mathcal{C}$ or $\mathcal{A} \cap \mathcal{C} = \emptyset$ scenarios, and a specified threat label $\hat{\theta}$. Specifically, we want to learn a function of the form: $\mathcal{G} : (\mathcal{A}, \hat{\theta}, z) \rightarrow \tau$, where z is a latent variable. The area $\mathcal{A}$ may correspond to any geographical region and is not limited to the areas flown by trajectories in $\mathcal{D}$. The goal is for $\mathcal{G}$ to generate drone trajectories over any region, while preserving threat-conditioned behaviors observed in $\mathcal{D}$. This capability is critical for building security warning systems in regions lacking comprehensive drone traffic data.

3.4. Methodology

STATE has 3 modules. The *Data Representation Module* extracts relevant information $F_{\mathcal{A}}$ about the target geographical area $\mathcal{A}$ (e.g., no-fly zones, population density). The *Potential Waypoint Set Generator* uses a GAN-based approach to generate the planar projection of a synthetically generated trajectory $\hat{\tau}$, conditioned on the information $F_{\mathcal{A}}$ extracted above, the target threat class $\hat{\theta}$, and a latent noise vector z (cf. Figure 3.2). *STATE's key innovation is the integration of a pre-trained threat classifier that provides an auxiliary loss, ensuring that the generated planar trajectory $\hat{\tau}$ aligns with the threat class $\hat{\theta}$.* Finally, the *Trajectory Reconstruction Module* arranges the trajectory's planar projection $\hat{\tau}$ into a sequential trajectory τ and assigns altitude values to each waypoint. We describe these modules in detail below.

3.4.1. Data Representation Module

This module encodes the target area $\mathcal{A}$ into a multi-channel feature tensor $F_{\mathcal{A}} \in \mathbb{R}^{H \times W \times L}$, where H and W denote the spatial dimensions of the area, and L is the number of feature channels. Figure 3.2 shows the extracted tensor for part of The Hague. Channels include:
No-Fly Zone Map $F_{\mathcal{A}}^{NFZ} \in \{0,1\}^{H \times W}$: A binary image indicating the presence (1) or absence (0) of restricted airspace (e.g., airports, military bases).[2].
Population Density Map $F_{\mathcal{A}}^{PD} \in \mathbb{R}^{H \times W}$: A heatmap showing population density in different regions obtained in high-resolution from the Humanitarian Data Exchange[3].
Satellite Imagery $F_{\mathcal{A}}^{SI} \in \mathbb{R}^{H \times W \times 3}$: High-resolution RGB image from OpenStreetMap[4].

[2]This information is usually publicly available. For The Hague region, we collected 85 no-fly zones from https://map.godrone.nl
[3]https://data.humdata.org
[4]https://www.openstreetmap.org

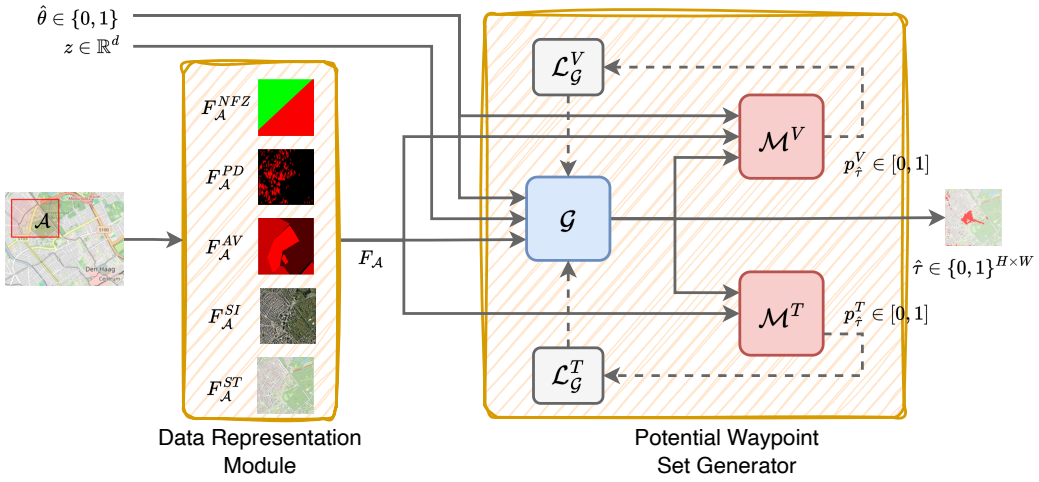


Figure 3.2. **STATE's Architecture**: The *Data Representation Module* represents the target geographical region $\mathcal{A}$ via a multi-channel feature tensor, including the *No-Fly Zone* Map $F_{\mathcal{A}}^{NFZ}$, the *Population Density Map* $F_{\mathcal{A}}^{PD}$, the *Satellite Imagery* $F_{\mathcal{A}}^{SI}$, the *Street Map* $F_{\mathcal{A}}^{ST}$, and the *Asset Value Map* $F_{\mathcal{A}}^{AV}$. Then, the *Potential Waypoint Set Generator* takes geographic features $F_{\mathcal{A}}$, the target threat class $\hat{\theta}$, and a noise vector z as input. It outputs a planar trajectory $\hat{\tau}$ that is evaluated by the Trajectory Validity Discriminator $\mathcal{M}^V$ network which distinguishes real from synthetic trajectories, and the Threat Alignment Network $\mathcal{M}^T$ that ensures consistency with the intended threat class. In this case, we are conditioning the generation process on the threatening class, i.e., $\hat{\theta} = 1$.

Street Map $F_{\mathcal{A}}^{ST} \in \mathbb{R}^{H \times W \times 3}$: A high-resolution RGB image of road networks and urban infrastructure from OpenStreetMap.

Asset Value Map $F_{\mathcal{A}}^{AV} \in \mathbb{R}^{H \times W}$: A heatmap (1-10 scale) quantifying the importance of parts of the target area. These values are annotated by local security experts and have been identified as a primary factor influencing threat perception (34). Two police officers from The Hague annotated relevant city areas.

Figure 3.2 shows an example of the feature maps extracted from the *Park Sorghvliet* region of The Hague and its adjacent districts. The *No-Fly Zone Map* $F_{\mathcal{A}}^{NFZ}$ shows that about half of this area (red region in Figure 3.2) is an NFZ. The *Population Density Map* $F_{\mathcal{A}}^{PD}$ shows high density in the city center, but low density in the park. The *Asset Value*

Map $F_{\mathcal{A}}^{AV}$ shows that security experts assigned a high threat level to the park, suggesting its relevance for security planning.

The complete representation $F_{\mathcal{A}}$ of the target area is formed by concatenating all channels along the depth dimension: $F_{\mathcal{A}} = \left[F_{\mathcal{A}}^{NFZ} \oplus F_{\mathcal{A}}^{PD} \oplus F_{\mathcal{A}}^{SI} \oplus F_{\mathcal{A}}^{ST} \oplus F_{\mathcal{A}}^{AV}\right], F_{\mathcal{A}} \in \mathbb{R}^{H \times W \times 9}$, where $\oplus$ denotes channel-wise concatenation. All channels are spatially aligned such that each pixel location corresponds to the same geographic coordinate in $\mathcal{A}$. For notational simplicity, we will refer to $F_{\mathcal{A}}$ as F henceforth.

3.4.2. Potential Waypoint Set Generator

This module generates the planar projection $\hat{\tau}$ of a trajectory τ, i.e., the unordered set of waypoints over the target geographical area $\mathcal{A}$. It has three main components: the *Waypoint Generator* network $\mathcal{G}$, the *Trajectory Validity Discriminator* $\mathcal{D}^V$, and the *Threat Alignment* classifier $\mathcal{D}^T$.

The Waypoint Generator uses an encoder-decoder architecture to synthesize $\hat{\tau}$ from the target geographic area features F, the target threat class $\hat{\theta}$, and a noise vector z. Formally, it computes the function:

$$\mathcal{G} : (F, \hat{\theta}, z) \rightarrow \hat{\tau} \in \{0, 1\}^{H \times W},$$

where $z \in \mathbb{R}^d$ is a latent d-dimensional vector sampled from a normal distribution, i.e., $z \sim \mathcal{N}(0, \mathbf{I}_d)$. It uses a CLIP-based encoder (98) to extract relevant spatial features from the target geographic area. The encoder transforms the multi-channel tensor $F \in \mathbb{R}^{H \times W \times 9}$ into a latent h-dimensional feature embedding $X_F \in \mathbb{R}^h$. Concurrently, a feed-forward

neural network encodes the target threat class $\hat{\theta}$ into a k-dimensional latent vector $X_{\hat{\theta}} \in \mathbb{R}^k$.

The target area's features X_F, the encoded threat class $X_{\hat{\theta}}$, and the latent noise vector z are concatenated to form a unified input vector $X' = \left[X_F \oplus X_{\hat{\theta}} \oplus z\right] \in \mathbb{R}^{h+k+d}$.

A decoder network then processes X' to generate the trajectory's planar projection $\hat{\tau} \in \{0,1\}^{H \times W}$, i.e. the decoder outputs an unordered set of waypoints which the synthetic trajectory might fly over.

Figure 3.2 provides a visual illustration of a generated waypoint set $\hat{\tau}$ over *Park Sorghvliet* in The Hague, corresponding to the geographical setting described in Section 3.4.1. The generation is conditioned on the presence of threat, i.e., $\hat{\theta} = 1$. For visualization purposes, the trajectory's planar projection $\hat{\tau}$ is overlaid (in red) on the corresponding street map. We note that the generator primarily selects waypoints in densely populated areas surrounding parks not covered by no-fly restrictions which is realistic. A smaller number of waypoints are also located within the park boundaries, which had been previously annotated as a high-value asset by police officers. This selection reflects the influence of the threat label on the generation process, guiding the model toward a trajectory that may pose threat from a security perspective.

The Trajectory Validity Discriminator $\mathcal{M}^V$ is a binary classifier which assesses whether $\hat{\tau}$ is real (i.e., sampled from dataset $\mathcal{D}$) or synthetic (i.e., generated by $\mathcal{G}$). It is trained jointly with the generator and is conditioned on the same inputs. Formally, it is defined as:

$$\mathcal{M}^V : (\hat{\tau}, F, \theta) \rightarrow p^V_{\hat{\tau}} \in [0,1],$$

where $p^V_{\hat{\tau}}$ represents the probability that $\hat{\tau}$ is a real trajectory. A higher value of $p^V_{\hat{\tau}}$ indicates that $\mathcal{M}^V$ has greater confidence in the authenticity of $\hat{\tau}$.

The Threat Alignment Network $\mathcal{M}^T$ is a pre-trained binary classifier that estimates the probability of the target threat class $\hat{\theta}$ for a given planar trajectory $\hat{\tau}$. Formally, it is defined as:

$$\mathcal{M}^T : (\hat{\tau}, F) \rightarrow p^T_{\hat{\tau}} = P(\hat{\theta}|\hat{\tau}, F) \in [0, 1],$$

Incorporating F is critical in assessing whether a trajectory is threatening or not because past work (34) has shown that locations of high-value assets on the ground play a critical role in determining if a trajectory is threatening or not. Note that $\mathcal{M}^T$ is pre-trained on real trajectories from $\mathcal{D}$ and remains fixed during adversarial training of the *Waypoint Generator* and *Trajectory Validity Discriminator*. Section 3.5.1.2 provides full implementation details on $\mathcal{G}$, $\mathcal{M}^V$ and $\mathcal{M}^T$.

3.4.2.1. Adversarial Training. The *Waypoint Generator* $\mathcal{G}$ and the *Trajectory Validity Discriminator* $\mathcal{M}^V$ are adversarially trained in a manner consistent with Conditional Generative Adversarial Networks (cGANs)(84). Specifically, the *Waypoint Generator* $\mathcal{G}$ is trained to synthesize planar trajectories $\hat{\tau}$ over F that not only resemble real-world planar trajectories but also align with the target threat class $\hat{\theta}$. Its loss function consists of two components:

Trajectory Validity Loss ensures that the generated trajectory $\hat{\tau}$ is realistic through feedback from the discriminator $\mathcal{M}^V$. The relative loss term is as follows:

$$\mathcal{L}^V_{\mathcal{G}} = \mathbb{E}_{\mathbf{z}}\Big[-\log p^V_{\hat{\tau}}\Big] = \mathbb{E}_{\mathbf{z}}\Big[-\log \mathcal{M}^V\big(\mathcal{G}(F, \hat{\theta}, z), \hat{\theta}, F\big)\Big]$$

Threat Alignment Loss encourages $\mathcal{G}$ to generate planar trajectories that are aligned with the target threat class $\hat{\theta}$. It incorporates the prediction $p^T_{\hat{\tau}}$ of the *Threat Alignment Network* $\mathcal{M}^T$ as follows:

$$\mathcal{L}^T_{\mathcal{G}} = \mathbb{E}_{\mathbf{z}}\Big[-\log p^T_{\hat{\tau}}\Big] = \mathbb{E}_{\mathbf{z}}\Big[-\log P(\mathcal{M}^T(\mathcal{G}(\mathcal{F}, \hat{\theta}, z)) = \hat{\theta})\Big]$$

The *Waypoint Generator*'s combined loss function is a linear combination of these terms: $\mathcal{L}_{\mathcal{G}} = \lambda_V \cdot \mathcal{L}^V_{\mathcal{G}} + \lambda_T \cdot \mathcal{L}^T_{\mathcal{G}}$, where λ_V and λ_T are scalar weights that control the relative importance of trajectory realism and threat alignment, respectively.

In the example in Figure 3.2, training $\mathcal{G}$ with a large λ_T value may lead to a generator that places a disproportionate number of waypoints within the no-fly zone (e.g., in *Park Sorghvliet*). While this may be consistent with the desired threat label $\hat{\theta}$, it conflicts with operational constraints and typical flight patterns, as real trajectories — regardless of threat intent - are unlikely to traverse extensively or exclusively through restricted airspace.

The *Trajectory Validity Discriminator* $\mathcal{M}^V$ is trained jointly with $\mathcal{G}$ to separate real planar trajectories η sampled from $\mathcal{D}$ and synthetic planar trajectories $\hat{\tau}$ generated by $\mathcal{G}$. Its loss is as follows:

$$\begin{aligned}\mathcal{L}_{\mathcal{M}^V} = \mathbb{E}_{\eta\sim\mathcal{D}}\Big[-\log \mathcal{M}^V(\eta, \hat{\theta}, F)\Big] + \\ \mathbb{E}_{\mathbf{z}}\Big[-\log\big(1 - \mathcal{M}^V\big(\hat{\tau}, \hat{\theta}, F\big)\big)\Big].\end{aligned}$$

Note that this loss does not incorporate feedback from the *Threat Alignment Network* as $\mathcal{D}^V$ is solely used to assess the realism of the generated planar trajectories without explicitly enforcing alignment with a particular threat class.

3.4.3. Trajectory Reconstruction Module

This module synthesizes a complete trajectory τ from its planar projection $\hat{\tau}$. This process includes (i) reconstructing a temporally ordered sequence of waypoints from $\hat{\tau}$, and (ii) assigning altitude values to τ's waypoints. This post-processing step is independent of the training phase of the *Potential Waypoint Set Generator*.

3.4.3.1. Temporal Sequencing. This process reconstructs a temporal trajectory τ from the planar projection $\hat{\tau} \in \{0,1\}^{H \times W}$, as illustrated in Figure 3.3. It starts by identifying the contour Ω of the largest connected component of waypoints in $\hat{\tau}$, denoted $\Omega = \{w_1, w_2, \ldots, w_l\}$. For example, this region represents the most extensive region of adjacent "red" pixels in $\hat{\tau}$ in Figure 3.3.

All waypoint pairs (w_s, w_e) from this set are considered: $w_s, w_e \in \Omega \wedge d(w_s, w_e) < \xi$ where d is Manhattan distance, and ξ is an empirically optimized threshold that governs the spatial proximity required for candidate trajectory formation. For each pair (w_s, w_e), a candidate trajectory $\pi = \{w_s, w_2, w_3, \ldots, w_e\}$ is constructed through a stochastic random walk connecting w_s to w_e[5].

The complete set of candidate trajectories, denoted $\Pi = \{\pi_1, \pi_2, \ldots, \pi_L\}$, where $L = \binom{l}{2}$ is the number of waypoint pairs. Figure 3.3 illustrates some candidates generated from the same planar projection over the *World Forum* convention center in The Hague. Although all candidates traverse the same region (*Park Sorghvliet* and its surrounding districts), they may vary significantly in spatial layout and flight dynamics. For this reason, each trajectory in Π is evaluated by the Threat Alignment Network $\mathcal{M}^T$ (cf.

[5]The existence of such a random walk is guaranteed, as both w_s and w_e belong to the same connected component.

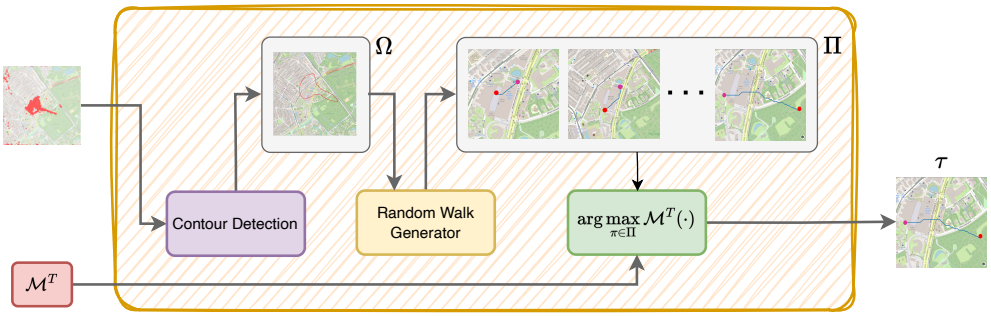


Figure 3.3. **Temporal Sequencing**: This module reconstructs a temporally ordered trajectory τ from the binary planar projection $\hat{\tau}$. It begins by identifying the contour Ω of the largest connected component in $\hat{\tau}$. All waypoint pairs (w_s, w_e) along the contour are used to generate candidate trajectories via stochastic random walks, such that w_s and w_e are the starting point (in purple) and ending point (in red) of the trajectory. Each candidate trajectory $\pi \in \Pi$ is then evaluated using the Threat Alignment Network $\mathcal{M}^T$ to identify the trajectory most aligned with the target threat class $\hat{\theta}$.

Section 3.4.2) to ensure alignment with the target threat class $\hat{\theta}$. The final trajectory selected is the one that maximizes the probability of the target threat class $\hat{\theta}$:

$$\tau = \arg\max_{\pi\in\Pi} P(\hat{\theta}|\hat{\pi}, F) = \arg\max_{\pi\in\Pi} \mathcal{M}^T(\hat{\pi}, F),$$

where $\mathcal{M}^T(\hat{\pi}, F)$ represents the probability of π belonging to the target threat class $\hat{\theta}$, conditioned on the geographical area's features F.

3.4.3.2. Altitude Profiling. determines the altitude of each waypoint w_i in the generated trajectory τ. Let $\mathcal{D}_{\hat{\theta}} = \{\tau_{\hat{\theta}}^{(1)}, \tau_{\hat{\theta}}^{(2)}, \cdots, \tau_{\hat{\theta}}^{(B)} | \tau_{\hat{\theta}}^{(i)} \in \mathcal{D}\}$ be the subset of B real trajectories with the target threat label $\hat{\theta}$. We estimate the altitude distribution $\mathcal{N}_{\hat{\theta}}^{j}$ of the j-th waypoint of all trajectories in $\mathcal{D}_{\hat{\theta}}$. Subsequently, the altitude of the j-th waypoint of the generated trajectory τ is sampled from the corresponding altitude distribution $\mathcal{N}_{\hat{\theta}}^{j}$.

3.5. Experiments

3.5.1. Experimental Settings

3.5.1.1. Dataset. We use an 8 month dataset of 349 real drone trajectories over The Hague, collected by the Dutch police using the *Senhive*[6] drone tracking platform (34). This system detects drones within a 25 km radius by intercepting communication signals (e.g., radio frequencies) and logs their flight coordinates (latitude, longitude, altitude) at sub-second intervals.

Three police officers annotated the *threat score* of each trajectory on a 1–10 scale, with an inter-annotator agreement of 0.416 (using Cohen's κ). Following past work (34), trajectories with scores ≥ 7 are considered *Threatening* — others are *Safe*.

Table 3.1 reports summary statistics of the drone trajectory dataset released by the DEWS team (34). The dataset includes information on average flight duration, covered distance, altitude, and speed. The dataset comprises a total of 349 trajectories collected from 18 distinct drones operating within the urban area of The Hague. Out of the 349 trajectories, 42 (approximately 12%) were labeled as *threatening*, while the remaining 307 (88%) were marked as *safe*.

Additionally, three Dutch police officers annotated the importance of 92 parts of The Hague on a 1-100 scale, which we use to build the *Asset Value Map*. Figure 3.4 shows the distribution of these annotations. Notably, 33 out of 92 regions (35%) were assigned a value greater than 80, indicating a substantial concentration of high-importance assets across the city.

[6] https://www.senhive.com

Attribute	Threat Label $\theta = 0$	$\theta = 1$
No. Drones	18	
Duration (s)	400.0 (202.4)	466.1 (203.5)
Distance (m)	881.5 (1772.0)	1628.9 (1443.3)
Altitude (m)	73.7 (59.1)	112.4 (69.3)
Speed (km/h)	8.36 (7.79)	14.1 (11.3)
No. Trajectories	307	42

Table 3.1. Summary of drone trajectory attributes for Safe ($\theta = 0$) and Threatening ($\theta = 1$) trajectories. Each entry reports the mean and standard deviation (in parentheses).

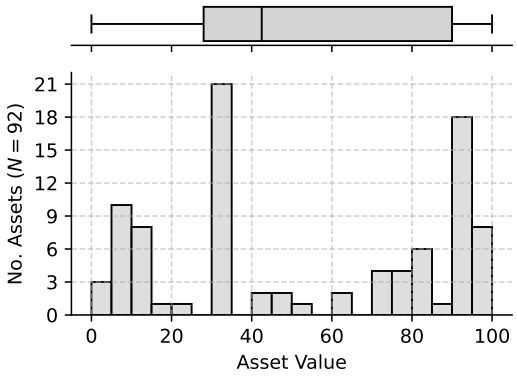


Figure 3.4. Distribution of values for the 92 assets annotated by three police officers.

3.5.1.2. Implementation Details. All experiments were conducted on a system with a 10th Gen Intel i9-10980XE processor, 256 GB of RAM, and an NVIDIA RTX A6000 with 48 GB memory. We used `Python 3.10` and `PyTorch 1.8` for our implementation.

Data Pre-Processing. For each trajectory $\tau \in \mathcal{D}$, we extract the corresponding bounding region from OpenStreetMap[7], applying a 200-meter padding around the flight path to ensure contextual information is preserved. We empirically determine that a resolution of 128×128 pixels offers a suitable trade-off between spatial detail and computational efficiency. Notably, increasing the resolution introduces a higher level of detail but also raises the complexity for the generator $\mathcal{G}$ to synthesize larger planar trajectories.

Waypoint Generator $\mathcal{G}$. We encode the target area's features via a pre-trained CLIP-based encoder (98) to project $F_{\mathcal{C}}$ into a latent representation $X_F \in \mathbb{R}^h$, where $h = 768$. The target threat label $\hat{\theta}$ is transformed into a latent vector $X_{\hat{\theta}} \in \mathbb{R}^k$, with $k = 8192$, using a fully-connected layer with ReLU activation. The size of the noise vector is set to $d = 100$. The combination of these vectors $X' \in \mathbb{R}^{768+8192+100}$ is projected into a vector

[7] https://www.openstreetmap.org

of size 8192 with one fully-connected layer with ReLU activation. The decoder processes this representation with five transposed convolution layers to generate the final planar trajectory $\hat{\tau} \in \{0,1\}^{128\times128}$.

Trajectory Validity Discriminator $\mathcal{M}^V$. This model is conditioned on the same inputs as the *Waypoint Generator* and outputs a binary classification decision. Its architecture has five convolution layers with LeakyReLU activation, followed by a sigmoid activation to produce the final classification probability. To enhance training stability and convergence speed, we use batch normalization between layers.

Adversarial Training. In each training iteration of *STATE*'s cGAN framework, we alternately update the discriminator $\mathcal{M}^V$ and the generator $\mathcal{G}$. The system is trained for 1000 epochs using a batch size of 64, using the Adam optimizer with a learning rate of $\eta_G = \eta_{D_V} = 10^{-4}$. The loss function of the *Waypoint Generator* is weighted using hyperparameters $\lambda_V = 0.6$ and $\lambda_T = 0.4$ to balance the objectives of trajectory realism and threat alignment.

Threat Alignment Network $\mathcal{M}^T$. The architecture of this component includes a CLIP-based encoder to embed the target area F and $\hat{\tau}$, followed by two fully-connected layers with ReLU activation function. $\mathcal{M}^T$ is pre-trained on real trajectories (34) and remains fixed during the adversarial training of the *Waypoint Generator*.

3.5.1.3. Metrics. We evaluate the quality of our framework by comparing synthetic trajectories with real ones. For each real trajectory $(\tau, \theta) \in \mathcal{D}$, we generate a corresponding synthetic trajectory $\tilde{\tau}$ over the same geographical area F_τ and with the same threat label θ. Let $\hat{P} = \{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_N\}$ be the set of planar projections of real trajectories,

and let $\hat{Q} = \{\hat{\tilde{\tau}}_1, \hat{\tilde{\tau}}_2, \ldots, \hat{\tilde{\tau}}_N\}$ be the set of planar projections of the corresponding synthetic trajectories. We assess the quality of the generated trajectories using the following metrics:

Distribution-Level Metrics. We analyze the statistical similarity between real and synthetic trajectories by comparing two distributions:

- *Trajectory Length Distribution*: We compare the empirical distributions of trajectory lengths, denoted as $\mathcal{L}_{\hat{P}}$ (real) and $\mathcal{L}_{\hat{Q}}$ (synthetic). Here, the trajectory length is defined as the number of waypoints that belong to the planar trajectory.
- *Asset-Visit Distribution*: We analyze the probability distribution of visits to assets of a given value v, denoted $\mathcal{V}_{\hat{P}}$ (real) and $\mathcal{V}_{\hat{Q}}$ (synthetic). This probability looks at all trajectories and divides the number of waypoints that visit assets with value v, by the total number of waypoints.

  To quantify the differences between these distributions, we use the Jensen-Shannon Divergence (JSD):

$$\text{JSD-TL} = D_{\text{JS}}(\mathcal{L}_{\hat{P}} \parallel \mathcal{L}_{\hat{Q}}),$$

$$\text{JSD-AV} = D_{\text{JS}}(\mathcal{V}_{\hat{P}} \parallel \mathcal{V}_{\hat{Q}}),$$

  where D_{JS} represents the Jensen-Shannon divergence (82).

Trajectory-Specific Metrics. We analyse spatial differences between real and synthetic trajectories with two metrics:

- *Pixel-wise difference.* We measure the pixel-wise difference MDE between the planar projection of a real trajectory $\hat{\tau} \in \hat{P}$ and its corresponding synthetic

trajectory $\hat{\tilde{\tau}} \in \hat{Q}$:

$$MDE(\hat{\tau}, \hat{\tilde{\tau}}) = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} |\hat{\tau}_{ij} - \hat{\tilde{\tau}}_{ij}|,$$

where $\hat{\tau}_{ij}$ and $\hat{\tilde{\tau}}_{ij}$ are the pixel values of the real and synthetic planar projections, respectively. This metric provides a localized comparison of trajectory accuracy.

- *Structural Similarity.* To assess the diversity of trajectories in a given set (either $\hat{P}$ or $\hat{Q}$), we measure the well-known *Structural Similarity Index* (SSIM) (129) among planar projections. A lower SSIM value indicates greater diversity, ensuring that the generated trajectories do not collapse into a limited set of similar patterns.

  Given two planar projections $\hat{\tau}_i, \hat{\tau}_j \in \hat{P}$, their SSIM score (130) is defined as:

$$SSIM(\hat{\tau}_i, \hat{\tau}_j) = \frac{(2\mu_{\tau_i}\mu_{\tau_j} + C_1)(2\sigma_{\tau_i\tau_j} + C_2)}{(\mu_{\tau_i}^2 + \mu_{\tau_j}^2 + C_1)(\sigma_{\tau_i}^2 + \sigma_{\tau_j}^2 + C_2)}$$

  where μ_{τ_i} and σ_{τ_i} denote the mean and variance of trajectory τ_i, $\sigma_{\tau_i\tau_j}$ represents their covariance, and C_1, C_2 are small constants to prevent division instability.

  The average structural similarity across all trajectory pairs $\hat{\tau}_i, \hat{\tau}_j \in \hat{P}$ is then given by:

$$SSIM_{\hat{P}} = \frac{1}{\binom{N}{2}} \sum_{i<j} SSIM(\hat{\tau}_i, \hat{\tau}_j),$$

  where $\binom{N}{2}$ is the number of unique trajectory pairs. We determine $SSIM_{\hat{Q}}$ by applying the same definition to the set of synthetic trajectories $\hat{Q}$.

3.5.2. Experimental Protocol

We designed four experiments to assess the quality and generalization abilities of *STATE*[8]:

- *Comparison with Baselines.* We evaluated *STATE* against two random baselines, i.e., *Random Walk* and *Monte Carlo Sampling*, and four recent trajectory generation approaches, i.e., *LSTM* (113), *VAE* (58), *Traj-GAN* (102), and *Diffusion-Synthesis* (140). For each real trajectory in the dataset, we generate a synthetic counterpart using both *STATE* and each baseline model, ensuring that generation occurs over the same geographical region and is conditioned on the same threat class. We then compute JSD-TL, JSD-AV, MDE, and $SSIM$, between real and synthetic trajectories. This experiment assesses the spatial fidelity of the generated trajectories with respect to real-world movement patterns, controlling for both spatial context and threat conditioning.
- *Ablation Study.* We evaluated the contribution of each module of *STATE*, namely, the *Threat Alignment Network* $\mathcal{M}^T$, the target geographical area $\mathcal{A}$, and the encoder to extract spatial features X_F.
- *Adversarial Training.* We analyze the training dynamics of *STATE* to validate that the generator $\mathcal{G}$ successfully learns to produce realistic and threat-aligned trajectories. We track the Jensen-Shannon Divergence between generated and real trajectory distributions across training epochs for both asset visitation frequency and trajectory length. Additionally, we examine the evolution of latent

[8]All hypotheses tested in this chapter report Bonferroni-corrected p-values, obtained with Mann-Whitney U-test (80), to adjust for multiple hypothesis testing.

embeddings produced by the *Trajectory Validity Discriminator* $\mathcal{M}^V$ at the beginning and end of training. This experiment empirically proves that the adversarial training process successfully aligns the generated distribution with real trajectory patterns.

- *Generalization to Unseen Regions.* We assess *STATE*'s ability to learn from trajectories on one region and generate trajectories for a different region. As we only knew police officials from The Hague, we partitioned the city into multiple sub-regions, treating each as a functionally independent area (analogous to distinct regions). *STATE* was trained on sub-regions that contained real trajectories, and then evaluated on the remaining sub-regions that had no recorded flights and are disjoint from the training data. This mirrors our use case for *STATE*: synthesizing plausible drone trajectories for regions where no flight data is available.

3.5.3. Comparison with Baselines

3.5.3.1. Baselines' Configuration. We evaluated *STATE* against six baselines configured as follows:

Random Walk: Given the target geographic region $F_{\mathcal{A}}$, we generate a synthetic trajectory by randomly sampling an initial waypoint from $\hat{\tau} \in \{0,1\}^{H\times W}$ and then iteratively sampling adjacent pixels the next waypoints.

Monte Carlo Sampling: This method estimates the distributions $\phi_{\theta=0}$ and $\phi_{\theta=1}$ of heading angles corresponding to both safe and threatening trajectories, respectively. The trajectory is initialized at a uniformly sampled waypoint within $\hat{\tau} \in \{0,1\}^{H\times W}$. But

subsequent waypoints are generated as follows: given the target threat label $\hat{\theta}$, the heading direction is sampled from the $\phi_{\hat{\theta}}$ distribution, while step length is drawn from a normal distribution, i.e., step $\sim \mathcal{N}(1, 0.2)$. This strategy conditions the trajectory evolution on the assigned threat class, introducing statistical regularities observed in real-world data while maintaining elements of stochasticity.

Long Short-Term Memory (LSTM): We adapt the flight trajectory generation approach proposed in (113) by training a LSTM-based model that predicts the next waypoint, given the trajectory history. For a given starting waypoint of a real trajectory, the model is trained to iteratively predict the next adjacent waypoint of the trajectory's planar projection, until a special end token is generated. A standard cross-entropy loss function is used to supervise the next-location predictions during training. At inference time, we provide the trained LSTM model with a randomly selected initial location within the target geographical area F. The model then iteratively predicts the next waypoint by selecting the most probable outcome at each step until the end token is generated.

Variational Autoencoder (VAE): We adapt the trajectory generation framework introduced in (58) by training two separate VAE models for generating safe and threatening trajectories. Each VAE is trained solely on real trajectories in $\mathcal{D}$ that corresponds to its assigned threat class. The output of the generation process is a trajectory's planar projection $\hat{\tau} \in \{0, 1\}^{H \times W}$. During training, we evaluate the performance of each VAE by measuring the MDE with respect to real trajectories. At inference time, we generate a synthetic trajectory by first selecting the appropriate VAE model based on the target threat class $\hat{\theta}$. We then sample from the latent space of the corresponding VAE and decode the sampled latent vector into $\hat{\tau}$. This strategy conditions trajectory generation

Table 3.2. Performance comparison between *STATE*, its variants, and baseline methods on safe and threatening trajectories. Lower values are better for all metrics. Best results are in bold, second-best are underlined, *** indicates statistical significance (Bonferroni-corrected p-value < 0.001).

Method	Threatening Trajectories				Safe Trajectories			
	MDE (↓)	*SSIM* (↓)	JSD-AV (↓)	JSD-TL (↓)	*MDE* (↓)	*SSIM* (↓)	JSD-AV (↓)	JSD-TL (↓)
Random Walk	17.380	0.9284	0.0065	0.0539	15.040	0.9529	0.0042	0.0249
Monte Carlo	15.400	0.9440	0.0051	0.0502	14.420	0.9602	0.0061	0.0227
LSTM (113)	5.2501	0.8820	0.0045	0.0265	3.1901	0.9065	0.0052	0.0167
VAE (58)	9.6102	0.9782	0.0025	0.0426	10.710	0.9399	0.0038	0.0187
Traj-GAN (102)	8.2873	0.8561	0.0040	0.0282	6.5236	0.8255	0.0043	0.0174
Diffusion-Synthesis (140)	15.213	0.701	0.0045	**0.0045**	14.643	0.6950	0.0052	0.0100
STATE (Ours)	**1.2703*****	**0.6614*****	**0.0010**	0.0050	**1.6200*****	0.6644	0.0020	0.0050
w/o F	8.640	0.8203	0.0020	0.0457	6.250	0.8439	**0.0017**	0.0222
w/o CLIP	13.34	0.6932	0.0011	0.0093	19.94	**0.5230*****	0.0019	**0.0035**
w/o $\mathcal{M}^T$	7.530	0.9145	0.0019	0.0074	4.950	0.9260	0.0023	0.0065

on both the target threat class and the target geographical region which the trajectory is supposed to fly over.

Traj-GAN: We adapt the trajectory generation framework introduced in (102). This method encodes key locations of the target geographical area, and trains a GAN-based network to generate trajectories that preserve user privacy.

Diffusion-Synthesis (38): We adapt the diffusion-based trajectory generation framework introduced in (140). This method follows the DDPM framework (56), progressively adding Gaussian noise to trajectory images over $T = 500$ timesteps with variance β_t increasing linearly from 10^{-4} to 0.02. A U-Net architecture with residual blocks and self-attention mechanisms learns to reverse this corruption, integrating CLIP features and threat labels through learned embeddings to condition the denoising process. The model is trained for 100 epochs with batch size 32 using the Adam optimizer at learning rate 2×10^{-4}, minimizing mean squared error between predicted and actual noise. At inference time, the model generates trajectories by starting from random noise and iteratively denoising conditioned on the target threat class $\hat{\theta}$ and geographical region F.

3.5.3.2. Results. Table 3.2 shows the performance of each method according to the MDE, $SSIM$, JSD-AV, and JSD-TL metrics, evaluated separately for safe and threatening trajectories.

Our results show that STATE *consistently outperforms all baselines across all metrics.* For instance, it achieves an MDE of 1.2703 for threatening trajectories and 1.6200 for safe ones. In contrast, the best-performing baseline, *LSTM* (113), obtains MDE scores of 3.1901 (threatening) and 5.2501 (safe). Therefore, STATE achieves relative improvements of 75.8% and 49.2%, respectively. These differences are statistically significant with Bonferroni-corrected $p-$value< 0.001.

As expected, *Random Walk* and *Monte Carlo* baselines exhibit the poorest performance across all metrics. Surprisingly, the diffusion-based *Diffusion-Synthesis* (140) performs comparably to these random baselines despite its higher complexity. This underperformance can be attributed to data scarcity: our 349 trajectories may suffice to capture coarse distributional properties (38), as evidenced by relatively low JSD-AV (0.0045 for threatening and 0.0052 for safe trajectories) and JSD-TL (0.0045 for threatening and 0.01 for safe trajectories), but they are not enough to learn sharp spatial details needed for precise trajectory generation.

The *LSTM*, *VAE* and Traj-GAN baselines are better, highlighting the importance of learning trajectory structure from data. *LSTM* usually outperforms *VAE* and *Traj-GAN*, likely because it models the sequential nature of trajectory generation explicitly, predicting one waypoint at a time based on prior context. In contrast, *VAE* treats the trajectory as a holistic object, generating the planar projection in a single decoding step, which may limit its ability to capture sequential dependencies.

However, none of *LSTM*, *VAE*, *Traj-GAN*, and *Diff-RNTraj* approaches match the performance of *STATE*. The superior results achieved by *STATE* suggest that the combination of adversarial training and spatial context modeling provides significant advantages in synthesizing realistic drone trajectories.

3.5.4. Ablation Study

3.5.4.1. STATE's variants. We evaluated three variants of *STATE* to identify the most important components:

STATE w/o $\mathcal{M}^T$: This configuration drops the *Threat Alignment Network* during training by setting the threat alignment loss weight $\lambda_T = 0$, causing training to solely use the *Trajectory Validity Discriminator*. This allows us to assess the contribution of $\mathcal{M}^T$ in producing trajectories that align with the target threat class $\hat{\theta}$.

STATE w/o F: Here, we disable the model's access to the spatial context of the target geographical area $\mathcal{A}$. The *Potential Waypoint Set Generator* is conditioned only on the target threat label $\hat{\theta}$, with no information about the physical environment in which the trajectory is to be generated. This configuration tests the importance of incorporating geographic features for generating contextually plausible and spatially coherent trajectories.

STATE w/o CLIP: This variant replaces the CLIP-based encoder used to extract map features X_F with a simpler, histogram-based encoding strategy (138). This baseline reduces the model's capacity to capture spatial correlations and high-level semantics from the map. Comparing this configuration with the original allows us to assess the benefits of

using a high-capacity visual encoder for conditioning the generation process on complex geographical environment.

3.5.4.2. Results. Table 3.2 shows the performance of each *STATE* configuration w.r.t. the MDE, $SSIM$, JSD-AV, and JSD-TL metrics, evaluated separately for safe and threatening trajectories. Focusing on the MDE metric, we note that all ablated variants exhibit a substantial performance drop compared to the full *STATE* setup. Removing the geographical context (*w/o* F) or disabling the *Threat Alignment Network* (*w/o* $\mathcal{M}^T$) significantly degrades performance, highlighting the critical role of both components. These findings suggest a synergistic effect, wherein both spatial awareness and semantic threat alignment are necessary for generating accurate trajectories.

Interestingly, replacing the CLIP-based encoder with a histogram-based feature extractor (*w/o CLIP*) leads to worse MDE performance than entirely removing the geographical area input (*w/o* F). This holds for both threat classes and suggests that an ineffective map representation can mislead the waypoint generation process. We hypothesize that this outcome stems from the complex nature of F, which deviates from typical visual imagery and thus challenges simple encoding strategies.

Turning to the $SSIM$ metric, we observe smaller differences among configurations. The full *STATE* performs best on threatening trajectories, while *STATE w/o CLIP* surprisingly outperforms other variants for safe ones. This behavior depends on the nature of $SSIM$, which evaluates the diversity of generated trajectories with the same technique rather than comparing synthetic and real trajectories. In other words, *STATE w/o CLIP* generates very diverse safe trajectories, but these trajectories may be far from real trajectories as evidenced by the much higher MDE score.

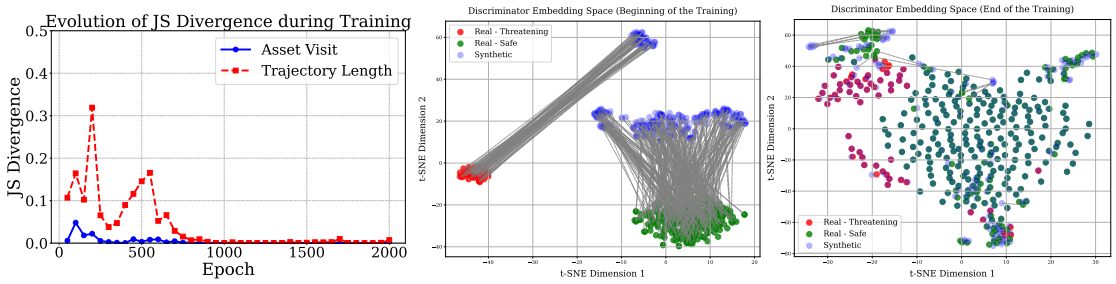
Finally, in terms of JSD-AV and JSD-TL metrics, differences between configurations are minimal, suggesting that while map encoding and threat alignment critically impact trajectory shape (MDE), they exert comparatively less influence on the high-level statistical properties, i.e., asset value and trajectory length distributions, of the generated trajectories.

3.5.5. Adversarial Training

Figure 3.5a shows the Jensen-Shannon Divergence (JSD) between the generated and real trajectory distributions across training epochs. Specifically, we report the JSD computed over two distributions: the asset visitation frequency and the trajectory length. In both cases, we observe a consistent decreasing trend, indicating that the generator increasingly aligns its output with real data as training progresses. Convergence is typically achieved around epoch 1000, which corresponds to our early stopping criterion.

To further investigate the training dynamics, we analyze the evolution of the latent embeddings produced by the *Trajectory Validity Discriminator* $\mathcal{M}^V$. Figures 3.5b and 3.5c show the two-dimensional t-SNE projections of these embeddings at the beginning (epoch 1) and the end of training, respectively. Specifically, for each real trajectory – either safe (in green) or threatening (in red) – we generate a synthetic counterpart (in blue), ensuring that generation occurs over the same geographical region and is conditioned on the same threat class. We use gray lines to connect a real trajectory with its synthetic counterpart.

Initially, $\mathcal{M}^V$ successfully maps real and synthetic trajectories to distinct regions of the embedding space (Figure 3.5b). In contrast, at the end of training (Figure 3.5c), we observe that real (green and red) and synthetic (blue) trajectories largely overlap in the



(a) (b) (c)

Figure 3.5. **Adversarial Training:** (a) Jensen-Shannon Divergence (JSD) between the generated and real trajectory distributions over training epochs, evaluated for both asset visit (blue) and trajectory length (red) distributions; (b-c) The t-SNE visualizations of the embeddings produced by $\mathcal{M}^V$ after the first training epoch (b) and at the end of training (c). Grey lines connect real trajectories with their synthetic counterparts.

embedding space. This indicates that the *Waypoint Generator* $\mathcal{G}$ has successfully learnt to produce realistic trajectories that are indistinguishable from real samples, thereby deceiving the discriminator $\mathcal{M}^V$.

In addition, we also observe that at the beginning of training (Figure 3.5b) $\mathcal{M}^V$ is able to distinguish real threatening trajectories (in red) and real safe trajectories (in green), without any supervision. This capability depends on the fact that $\mathcal{M}^V$ is conditioned on the target threat label $\hat{\theta}$, which provides explicit guidance regarding the semantic class of each input trajectory. Consequently, the corresponding synthetic trajectories (in blue) are also separated in the embedding space according to their assigned threat label. In other words, at the beginning of the training process, $\mathcal{M}^V$ is leveraging $\hat{\theta}$ to distinguish between real and synthetic samples.

Conversely, at the end of training (Figure 3.5c), the embeddings of threatening (red) and safe (green) trajectories are no longer clearly separable. This suggests that $\mathcal{M}^V$, while initially leveraging the threat label for classification, has ultimately prioritized features that do not depend on threat semantics to distinguish real from synthetic trajectories.

Table 3.3. **Expert Evaluation**: Post-hoc assessment of synthetic trajectories generated with *STATE* and the VAE baseline.

Method	Precision		Recall		F1-score		Accuracy
	$\theta = 0$	$\theta = 1$	$\theta = 0$	$\theta = 1$	$\theta = 0$	$\theta = 1$	
VAE (58)	**0.970**	0.375	0.767	**0.857**	0.857	0.522	0.780
STATE	0.878	**0.733**	**0.900**	0.687	**0.888**	**0.709**	**0.839**
Δ	−9.48%	95.5%	17.34%	−19.8%	3.62%	35.8%	7.15%

In other words, the conditioning on $\hat{\theta}$ serves as auxiliary information during training, but it does not dominate the embedding space learnt by the discriminator $\mathcal{M}^V$ once the generator becomes proficient. This observation supports our design choice of incorporating a separate threat-aware feedback mechanism via the *Threat Alignment Network* $\mathcal{M}^T$, which explicitly enforces alignment between the generated trajectory and the intended threat class. By decoupling the semantic alignment objective from the adversarial realism objective, we ensure that both trajectory plausibility and threat specificity are jointly optimized by the *Waypoint Generator* $\mathcal{G}$.

3.5.6. Generalization to Unseen Regions

We generated 100 synthetic trajectories (50 safe and 50 threatening) using *STATE* and another 100 using the VAE baseline[9], over unseen region of The Hague. Two police officers from The Hague independently annotated the perceived threat level of each synthetic trajectory. We use this "ground truth" to compute standard performance metrics, i.e., accuracy, precision, recall and F1-score.

3.5.6.1. Results. Table 3.3 shows class-wise precision, recall, and F1-score for *STATE* and VAE. *STATE consistently outperforms VAE across most metrics.* For instance, on the

[9]We selected the VAE baseline because it is conditioned on both the target geographical area and the intended threat class, in contrast to the LSTM baseline, which conditions only on location.

threatening class ($\theta = 1$), it achieves an F1-score of 0.709, representing a 35.8% improvement over VAE. The performance gap is primarily due to VAE's low precision (0.375) on threatening trajectories, while *STATE* attains 0.733, marking a relative improvement of 95.5%.

In addition, we observe a consistent performance disparity between safe ($\theta = 0$) and threatening ($\theta = 1$) classes. VAE achieves an F1-score of 0.857 for safe trajectories and 0.522 for threatening ones. Similarly, *STATE* reaches 0.888 and 0.709 for safe and threatening trajectories, respectively. This asymmetry likely reflects the class imbalance present in the original DEWS dataset – 307 safe versus only 42 threatening real trajectories – which leads to a stronger optimization signal for the safe class during training. However, the performance gap between classes is narrower for *STATE*. We attribute this to the role of the $\mathcal{M}^T$, which provides an explicit supervision signal to the generator by penalizing threat-class misalignment. This design encourages the generator to maintain threat consistency even for underrepresented categories, i.e., threatening trajectories.

3.6. Conclusions, Limitations, & Future Work

We introduce *STATE* (*Safe and Threatening Adversarial Trajectory Encoder*), a novel cGAN-based framework for synthesizing drone trajectories conditioned on a target threat level. *STATE* generates both safe and threatening trajectories over arbitrary geographical areas, even in the absence of real-world flight data. We show that *STATE* outperforms five trajectory generation baselines in terms of spatial plausibility and threat alignment. To our knowledge, this is the first system that generates threat-conditioned drone trajectories for security applications. But our work has limitations.

Geographical Scope. All experiments were conducted using data from The Hague, in the Netherlands. While this is realistic, recruiting the very small number of police experts in other geographies proved challenging. This remains important future work.

Threat Detection. *STATE* relies on a pre-trained *Threat Alignment Network* ($\mathcal{M}^T$) to align synthetic trajectories with the target threat class. $\mathcal{M}^T$ exhibits lower performance on threatening trajectories. But threat classification is not the primary contribution of this work, and our framework remains agnostic to the specific threat classifier used.

Threat Semantics. We adopt a binary threat classification schema. However, threat perception in operational contexts often involves nuanced attributes such as intent, proximity to critical assets, or temporal factors. Future work could explore multi-dimensional threat representations to provide more granular control and interpretability over the generation process.

Dual-Use Considerations. We acknowledge that *STATE* has dual-use potential and could be misused for malicious drone planning. However, security officials deemed the benefits (e.g., improved drone threat defense) greater than the risks and approved the release of the chapter, data, and algorithms.

CHAPTER 4

Declarative Logic-based Pareto-Optimal Agent Decision Making

Chapters 2 and 3 established capabilities for identifying threatening drone trajectories and generating synthetic training data. However, threat identification alone does not determine appropriate defensive responses. Autonomous defensive systems operating in civilian environments must satisfy legal, regulatory, and ethical constraints while pursuing operational objectives. There are many applications where an autonomous agent, e.g., a drone, can simultaneously perform many sets of actions. It must choose one set of actions based on some behavioral constraints on the agent. Past work has used deontic logic to *declaratively* express such constraints in logic, and developed the concept of a feasible status set (FSS), a set of actions that satisfy these constraints. However, multiple FSSs may exist and an agent needs to choose one in order to act. As there may be many different objective functions to evaluate status sets, we propose the novel concept of *Pareto-optimal feasible status sets* or POSS. We show that checking if a status set is a POSS is co-NP-hard. We develop an algorithm to find a POSS and in special cases when the objective functions are monotonic (or anti-monotonic), we further develop more efficient algorithms. Finally, we conduct experiments to show the efficacy of our approach and we discuss possible ways to handle multiple Pareto-optimal Status Sets.

4.1. Introduction

Autonomous agents are becoming increasingly important in the real-world. A good example is self-driving cars (SDC for short) where agents already control several functions, such as lane changes and speed changes in Tesla vehicles (39). Another example involves proposals for nuclear power plants involving agents that can increase coolant pressure, temperature, and more (66). Autonomous agents are also being proposed for use with implantable medical devices (40). These are critical applications. They are characterized by certain common features:

Declarative Operating Rules. The agents involved need to take actions while respecting declaratively specified behavioral requirements, i.e., the desired behavior should be specified in an easy to understand high-level language such as logic, not code specifying how that desired behavior is to be accomplished (16). For instance, a self-driving car should be forbidden to move into a lane when the location it is moving to is going to be occupied by another vehicle. It may be obligatory for an autonomous agent to shut off certain processes when the coolant level in a power plant drops below some threshold. An agent managing an implantable device may be permitted but not obliged to warn the user when there is a danger of a non-life threatening malfunction. All such behavioral requirements should be stated in a declarative language that is easy to understand for domain experts.

Concurrent Actions. The agents may perform zero, one or more actions simultaneously, e.g., shut off a process, send messages to other agents and/or human users.

Constraints on Actions. There are constraints on sets of actions that can be done concurrently, e.g., coolant pressure cannot be increased and decreased at the same time.

Certain combinations of actions may lead to impossible or undesirable states (e.g., one where there is a nuclear leak). Such constraints can be expressed easily in high-level logical languages.

Autonomy. The agents are autonomous, i.e., they can make a conscious choice between different sets of actions that they can take at a given time.

Multiple Objectives. The agents may measure the desirability of a set of actions along multiple dimensions, e.g., annoyance to user if she gets too many alerts, maximizing safety of the environment considered, cost, time, and more.

Deontic logic (48; 90) has been studied for more than 50 years. It extends classical logic to support reasoning with the effects of actions on the state of the world. In multi-agent applications, agents should operate under certain behavioral constraints. In self-driving cars, for instance, agents should obey the rules of the road. They may be permitted to do certain things in some conditions, forbidden from doing things in other conditions, obliged to do some things in yet other circumstances, and more. Deontic logic therefore studies the permissions, obligations, and forbidden modalities and develops the logical foundations of their interactions both with each other, with classical logic and actions.

A declarative deontic logic framework within which we can express what the agent is permitted to do, obliged to do, and forbidden to do in various situations has already been proposed by (44; 45). Their "IMPACT" framework defines "agent programs" that encode desired declarative agent behaviors, the syntactic concept of a status set, and the semantic concept of a feasible status set (FSS). Intuitively, an FSS captures a set of actions that the agent can perform, compatible with its operating rules, constraints on actions, concurrency constraints, and the deontic logic modalities. IMPACT was shown

in (118) to support easy articulation of desired high-level behavioral requirements for 3 broad application areas: transportation, supply chain management, and an online store. However, IMPACT does not incorporate any objective functions. Subsequently, (116) proposed the concept of optimal status sets in which an agent can choose a feasible status set (and hence a set of actions to perform) that optimizes a single objective function, but multiple objective functions are not allowed.

Real world agents may consider many factors. A nuclear power monitoring agent may wish to minimize the number of alerts sent to the engineering team while simultaneously maximizing safety. This requires consideration of two orthogonal but incomparable objective functions. In general, no single solution might simultaneously optimize all objective functions. A typical approach to deal with this is *Pareto-optimality* (95): a solution is *Pareto-dominated* if there is another solution that strictly improves some objective function value without degrading the other objective functions' values. A solution is *Pareto-optimal* if it is not Pareto-dominated. The *Pareto frontier* is the set of all Pareto-optimal solutions, all of which are considered equally good. To the best of our knowledge, the combination of deontic logic methods and Pareto-optimality has not been studied before.

In this chapter, we combine deontic logic (105) and Pareto-optimality. Specifically, we make the following contributions:

(1) We propose the new concept of a Pareto-Optimal (Feasible) Status Set, or POSS for short, which combines deontic logic and Pareto optimality. It combines the power of logic and the power of optimization. We show that the problem of checking if a given status set is Pareto-optimal is co-NP-hard, and it is co-NP-complete under some reasonable assumptions.

(2) We develop the first algorithm to find a POSS for a given agent-state pair.

(3) We develop the first algorithms to compute POSS's when the objective functions are monotonic (or anti-monotonic).

(4) We report on a prototype implementation of our framework, showing that POSS works well on a realistic collaborative SDC scenario, where we vary several parameters and assess their impact on performance.

The chapter is organized as follows. Section 4.2 discusses related work. Section 4.3 provides a motivating example of a futuristic collaborative SDC scenario in which multiple cars collaborate to achieve their objectives. Section 4.4 provides a brief overview of IMPACT (44; 45; 118). Section 4.5 extends IMPACT so that agents consider multiple objective functions and introduces Pareto-optimal (Feasible) Status Sets. It then studies the complexity of the problem. Section 4.6 presents exact and heuristic algorithms to solve this problem. Section 4.7 presents an experimental assessment of these algorithms. Section 4.8 discusses possible ways to handle the situation where a Pareto front has multiple Pareto-optimal Status Sets. Section 4.9 describes limitations and outlook for future work. Section 4.10 concludes the chapter.

4.2. Related Work

We build upon deontic logic based agents introduced by (44; 118). While there is plenty of previous work on multiagent systems (e.g., see (21; 112; 54)), to the best of our knowledge, there is only one effort (116) that tries to build agents that optimize their actions in the presence of both deontic behavioral rules and constraints. (116) is limited to one objective function, while our approach can handle several. (55) proposes the

jDALMAS system, which includes a preference structure based on a theory of normative positions (69). They consider a partial ordering on actions to be taken by an agent, but do not consider explicit *numerical* objective functions. (55) does not consider objective functions. In addition, we develop novel algorithms for weakly/strongly monotonic and anti-monotonic objective functions, whereas neither (116) or (55) consider such specialized objective functions.

(22) provides an excellent overview of logic-based agent systems, but does not say much about deontic logic (except for the jDALMAS effort mentioned above) or optimization, suggesting that there is a lot of room for work in this space.

There have been many numeric approaches to Pareto optimization (70; 67; 74; 75; 136; 33) that do not involve logic. All of these algorithms focus on searching for optimal solutions over the feasible solution space, but they do not consider how to generate feasible solutions over a *logical* solution space, which is fundamental for the logical approach. In multi-agent settings, (26) proposes a distributed approach to find a Pareto-optimal solution. (127) looks at a very specific scheduling problem where two agents compete to work on a machine: one agent tries to minimize the number of delayed jobs it initiated, while the other agent wants to maximize a different quantity associated with its jobs. (137) studies a similar situation. (27) combines deterministic policy gradients with Pareto optimization to develop good recommender systems. (128) provides an excellent view of agent-based methods for network traffic management. While these are important efforts, none of them combine logic and optimization. The behaviors of these agents are not declaratively specified and in some cases, optimization focuses on very specific

objective functions. In contrast, we provide declarative deontic logic based constraints[1] that are easy to explain to stakeholders and show how our objective functions can be easily optimized. We present different types of algorithms depending on the different properties of the objective functions (e.g., no restrictions on the objective function, weakly/strongly monotonic, and weakly/strongly anti-monotonic). Additionally, we propose approximation algorithms.

Future work could examine the use of probabilistic and/or defeasible deontic rules in situations where there is uncertainty about the state and/or where there is uncertainty in whether certain behavioral norms can be relaxed (32; 89).

4.3. Motivating Example

Consider a divided highway as shown in Figure 4.1. Cars are traveling from left to right on one side of the highway which can be thought of as a matrix. For simplicity, in this example, the number of cars is fixed. Some cells are marked with an "X" to indicate that there is no road there. Some cells are marked "EXIT" to specify that there is an exit at that location. The exit also shows the destination (location A or B). A car that exits at location $(4, 4)$ can make it to both locations A and B, while one that exits at $(4, 8)$

[1]A logical *theory* consists of a set of formulas (which include rules) in logic. An *interpretation* is an assignment of truth values to atomic formulas. A *model* of a logical theory is an interpretation that satisfies all the formulas in the logical theory. We can therefore see an analogy between integer 0-1 constraints and logic. Just as numeric 0-1 constraints, such as $x_a + x_b \geq 1$, constrain the space of solutions, logical formulas (including rules) constrain the space of interpretations that can be models. For instance, considering the logical formula $(a \vee b)$, the models are the interpretations that make at least one of a, b true. With the rule $a \rightarrow b$ acting as a constraint on the space of interpretations, we limit interest to those interpretations that either make b true or a false, or both. The articulation of how logical formulas and rules can be viewed as constraints goes back several decades. We refer the reader to (10; 11; 23) for a detailed exposition on why logical rules can be viewed as constraints. That said, not all constraints can be viewed as logical rules.

	1	2	3	4	5	6	7	8	9
1									
2									
3									
4	X	X	X	EXIT: A, B	X	X	X	EXIT: B	X

Figure 4.1. A highway represented as a matrix (cars traveling from left to right).

can only get to B. Initially, the red car is traveling at 2 cells/second, while the green and orange cars are traveling at 1 cell/second.

4.3.1. State

We assume the existence of an arbitrary but fixed logical language within which the state can be expressed. We assume readers are familiar with standard expressions such as constants, variables, predicate symbols, atoms, and formulas in logic (72). Following Prolog convention (28), we denote variables with upper case symbols—everything else will be denoted via lower case symbols.

At any point t in time, the *state* is a set of ground (i.e., containing constants only) logical atoms. In our motivating example, we use atoms of the following form:

(1) $at(car, x, y, t)$ describes the location (x, y) of a car at time t, e.g., $at(red, 1, 1, 1)$ says that at time 1, the red car is at location $(1, 1)$.

(2) $speed(car, s, t)$ is the speed of a car at time t, e.g., $speed(red, 2, 1)$ says that at time 1, the red car is traveling at 2 cells/second.

(3) $dest(car, loc)$ specifies the destination of a car, e.g., $dest(red, B)$ says that the red car's destination is B. This means the red car can take either exit in poss:fig:cars.

(4) $exit(y, loc)$ specifies where there is an exit and the location it leads to, e.g., $exit(8, B)$ says that there is an exit to B at location $(4, 8)$ (for simplicity, in this example, we assume exits are always in the bottom lane which is why the x value is not explicitly stated).

The table below shows the initial state S_0 of our running example—additionally, the initial state stores information on two exits at locations $(4, 4)$ and $(4, 8)$ leading to A, B and B, respectively. All three agents know this initial state.

car	*at*	*speed*	*dest*
red	(1,1)	2	B
green	(2,2)	1	A
orange	(3,2)	1	B

Furthermore, we assume the existence of a derived predicate $pred\_at(car, x, y, t + t')$ that predicts the location (x, y) of *car* at time $t+t'$, assuming inertia, i.e., that the car continues at its current speed without making any changes. This predicate can be readily derived from the *at* and *speed* predicates.

4.3.2. Agent Actions

We assume the existence of a language with a set of *action symbols*, which generate *action atoms* (or simply actions) using the constants and variables from the language used to express a state above. In our motivating example, the cars are capable of taking the following actions:

(1) $accel(car, s_1, s_2, t)$ says *car* accelerates from speed s_1 to s_2 at time t. Here $s_1 < s_2$.

(2) $decel(car, s_1, s_2, t)$ says *car* decelerates from speed s_1 to s_2 at time t. Here $s_1 > s_2$.

(3) $continue(car, t)$ keeps *car* going at its current speed at time t. So if the red car executes the action $continue(red, 1)$ at time 1, it will end up at location $(1, 3)$ at time 2.

(4) $go\_left(car, t)$ moves *car* one lane to the left. So if the green car performs this action in its initial state, then it will end up at time 2 at $(1, 3)$ (which would lead to a collision if the red car performed the action in the preceding bullet).

(5) $go\_right(car, t)$ moves *car* one lane to the right. So if the green car performs this action in its initial state, then it will end up at time 2 at $(3, 3)$ (which would lead to a collision with the orange car if that car were to execute the "continue" action at time 1).

(6) $exit(car, x, y, t)$ says *car* is going to exit the highway at location (x, y) at time t.

(7) $req(car1, car2, action, t)$ says that *car1* requests *car2* for permission to perform *action* at time t. For instance, $req(green, red, go\_left(green, 2, 2, 1, 1), 1)$ has green telling red that it would like to shift lanes to the left at time 1 from location $(2, 2)$ going to a current speed of 1. This is like a turn signal. But green can perform this action only if red responds that it will slow down or shift to the right in order to avoid a collision.

(8) $ok(car1, car2, action, t)$. Here *car2* agrees to the request by *car1* to perform *action* at time t.

(9) $deny(car1, car2, action, t)$ is the opposite situation: *car2* does not agree to the request by *car1* to perform *action* at time t.

Assumption. Without loss of generality, we assume that one tick of time is enough for a car to make a request, receive a response, and take an action.[2]

Each action α has a precondition $Pre(\alpha)$ which is a logical condition, an add list $Add(\alpha)$, and a delete list $Del(\alpha)$, both of which are sets of ground atoms. Action α is *executable* in state S_t if $Pre(\alpha)$ is true in S_t—if it is executed, then $Del(\alpha)$ is deleted from S_t while $Add(\alpha)$ is added to S_t in order to yield the new state.

As an example, for the action $\alpha = accel(car, s_1, s_2, t)$, we have $Pre(\alpha) = speed(car, s_1, t)$ & $(s_1 < s_2)$, $Del(\alpha) = speed(car, s_1, t)$, and $Add(\alpha) = speed(car, s_2, t+1)$.

Autonomy. Cars can make decisions autonomously. One car may deny (or not respond) to a request from another car.

Collaboration. The messaging actions $(req, ok, deny)$ enable agents to collaborate.

In general, we assume that an application domain has an associated set of action symbols and that we can define a notion of (ground) action atoms in the usual way (118; 30; 2). The above shows a specific set of action symbols and action atoms in our running SDC example.

4.4. Background: IMPACT Agents

We assume that arbitrary but fixed sets of actions and predicate symbols describing the state have been chosen as illustrated via the SDC example in the preceding section.

[2]One time unit t can be thought as having three parts: by $(t+0.33)$, a car sends one or more messages to other cars, by $(t+0.67)$ it receives responses, and it decides what to do before $(t+1)$ and does it exactly at $(t+1)$.

4.4.1. Agent Program

Every agent has an associated "agent program" that governs what the agent can and cannot do. In this section, we recall these definitions from (118). If α is an action, then $\mathbf{F}\alpha$, $\mathbf{P}\alpha$, $\mathbf{O}\alpha$, $\mathbf{Do}\alpha$ are *status atoms* indicating that an action is forbidden, permitted, obligatory, and to be done, respectively.

An *operating rule* (or just rule) is an expression of the form

$$SA \leftarrow \chi \ \& \ SA_1 \ \& \ \ldots \ \& \ SA_n$$

where $SA, SA_1, \ldots, SA_n$ are status atoms and χ is a logical condition (expressed using the predicate symbols). Intuitively, this rule says that if χ is true in the current state and if status atoms $SA_1, \ldots, SA_n$ are all true, then SA must also be true. These rules impose constraints—for example, the rule $\mathbf{F}\alpha \leftarrow \mathbf{Do}\beta$ imposes the logical constraint that if action β is done, then action α is forbidden.

An *agent program* is a finite set of rules.

Example 4.1. *The red car's allowed behavior can be expressed by the rules reported in poss:fig:agent-program.*

The first seven rules say that the red car is allowed to have a speed in the range $[1, 3]$. *This is a logical constraint which ensures that the red car cannot have a speed outside such a range. The next two rules say that the red car can take either of the two exits on the highway (as both lead to its destination, B) when it is near the exits. The following four rules say the car cannot go left from the leftmost lane, nor can it go right from the rightmost lane (exit action is not considered a right turn but a different action), while it*

$$
\begin{array}{rcl}
\boldsymbol{P}accel(red, S1, S2, T) & \leftarrow & 1 \leq S2 \leq 3. \\
\boldsymbol{P}continue(red, T) & \leftarrow & speed(red, S, T) \,\&\, 1 \leq S \leq 3. \\
\boldsymbol{P}decel(red, S1, S2, T) & \leftarrow & 1 \leq S2 \leq 3. \\
\boldsymbol{F}accel(red, S1, S2, T) & \leftarrow & S2 > 3. \\
\boldsymbol{F}decel(red, S1, S2, T) & \leftarrow & S2 < 1. \\
\boldsymbol{F}continue(red, T) & \leftarrow & speed(red, S, T) \,\&\, S > 3. \\
\boldsymbol{F}continue(red, T) & \leftarrow & speed(red, S, T) \,\&\, S < 1. \\
\boldsymbol{P}exit(red, 4, 4, T) & \leftarrow & at(red, 3, 4, T). \\
\boldsymbol{P}exit(red, 4, 8, T) & \leftarrow & at(red, 3, 8, T). \\
\boldsymbol{F}go\_left(red, T) & \leftarrow & at(red, X, Y, T) \,\&\, X = 1. \\
\boldsymbol{F}go\_right(red, T) & \leftarrow & at(red, X, Y, T) \,\&\, X = 3. \\
\boldsymbol{P}go\_left(red, T) & \leftarrow & at(red, X, Y, T) \,\&\, X > 1. \\
\boldsymbol{P}go\_right(red, T) & \leftarrow & at(red, X, Y, T) \,\&\, X < 3. \\
\boldsymbol{O}deny(Car1, red, & & \\
go\_left(Car1, X, Y, S, T), T) & \leftarrow & pred\_at(red, X', Y', T+1) \,\&\, \\
 & & pred\_at(Car1, X', Y', T+1) \,\&\, \\
 & & \boldsymbol{Do}req(Car1, red, \\
 & & go\_left(Car1, X, Y, S, T), T).
\end{array}
$$

Figure 4.2. Red car's agent program.

is permitted to go left (resp., right) when there is a lane on the left (resp., right). The last rule for the red car exhibits selfish behavior. It always denies requests that cause it to change its current behavior. All of these rules thus operate as logical constraints on actions.

The agent program for the green car is identical to that of the red car except for three differences: (i) it cannot reach a speed greater than 2, (ii) it is obliged to take the first possible exit, and (iii) the last rule makes the green car's behavior kinder and more cooperative as it is willing to adjust its own behavior when other cars request a move.

The agent program for the orange car is identical to that of the green car but it must stick to a constant speed of 1 and it is permitted to exit at either of the two exits.

An agent program specifies constraints on the agent's behavior: what the agent is obliged to do or forbidden from doing in certain situations and what it is permitted but

not required to do. Of course, the precondition of any permitted action must be true in a given state. Thus, these rules act as logical constraints on the agent's behavior.

4.4.2. Concurrent Action

An agent might choose to simultaneously do multiple things in a given state (e.g., a car may both accelerate and change lanes at the same time). In this case, we define a function called $\mathsf{conc}(A, S_t)$ which takes a set of actions A and state S_t as input and returns a new state S_{t+1}. (118) defines multiple possible ways of defining concurrent action execution.

4.4.3. Integrity Constraints

We can also write a set of integrity constraints defining valid states. Agents must not to take actions which would lead to a state that violates the integrity constraints. For instance, we would like an integrity constraint which says that an agent must not enter the same place as another agent. In general, an integrity constraint is either a *denial constraint* or a *definite constraint*, which we define below. If $A_1, \ldots, A_n$ are atoms (including atoms involving comparison operators), then a denial constraint has the form

$$\leftarrow A_1 \,\&\, \cdots \,\&\, A_n.$$

This denial constraint says that not all of $A_1, \ldots, A_n$ can be true in a given state. For example,

$$\leftarrow at(Car1, X, Y, T) \,\&\, at(Car2, X, Y, T) \,\&\, Car1 \neq Car2$$

is a denial constraint that says that two different cars cannot be in the same place at the same time (as this would be a collision). Many other denial constraints can be written

for our sample SDC scenario. Again, these are all logical constraints on what can and cannot be done in a given state.

If $A_0, A_1, \ldots, A_n$ are atoms (atoms involving comparison operators are also allowed), then a *definite constraint* is an expression of the form

$$A_0 \leftarrow A_1 \,\&\, \cdots \,\&\, A_n.$$

Intuitively, a definite constraint says that if $A_1, \ldots, A_n$ are all true in a given state, then A_0 must also be true in that state. For example, the definite constraint $Loc1 = Loc2 \leftarrow dest(Car, Loc1) \,\&\, dest(Car, Loc2)$ says that a given car has only one destination.

4.4.4. Action Constraints

Finally, we allow the specification of a form of logical constraints called action constraints with the same syntax of the integrity constraints previously introduced, but involving action atoms instead of ordinary atoms. For instance, in our SDC scenario,

$$\leftarrow go\_left(Car, T) \,\&\, go\_right(Car, T)$$

says that a car cannot try to move both left and right at the same time.

$$\leftarrow accel(Car, S1, S2, T) \,\&\, decel(Car, S1', S2', T)$$

says it cannot both accelerate and decelerate at the same time, and

$$\leftarrow ok(Car1, Car2, Action, T) \,\&\, deny(Car1, Car2, Action, T)$$

says it cannot both OK and deny the same request.

4.4.5. Status Set Semantics

In this section, we describe the semantics of agent programs from (44). A *status set SS* is a finite set of ground status atoms. There are many status sets that can be consistent with a given state and a given agent program. We call such status sets *feasible* and they are defined as follows.

Definition 4.1. *A status set SS is* feasible *w.r.t. a state S_t, an agent program P, a set of integrity constraints IC, and a set of action constraints AC, iff:*

(1) $\boldsymbol{O}\alpha \in SS \rightarrow \boldsymbol{P}\alpha \in SS$;

(2) $\boldsymbol{O}\alpha \in SS \rightarrow \boldsymbol{Do}\alpha \in SS$;

(3) $\boldsymbol{Do}\alpha \in SS \rightarrow \boldsymbol{P}\alpha \in SS$;

(4) $\boldsymbol{P}\alpha \in SS \rightarrow \boldsymbol{F}\alpha \notin SS$;

(5) $\boldsymbol{P}\alpha \in SS \rightarrow Pre(\alpha)$ *is true in* S_t;

(6) If $SA \leftarrow \chi \ \& \ SA_1 \ \& \ \ldots \ \& \ SA_n$ *is a ground instance of an operating rule in the agent program P and χ is true in state S_t and* $\{SA_1, \ldots, SA_n\} \subseteq SS$, *then* $SA \in SS$.

(7) $\{\alpha \mid \boldsymbol{Do}\alpha \in SS\}$ *satisfies the action constraints in AC;*

(8) If S_t satisfies IC, then the new state $\mathsf{conc}(\{\alpha \mid \boldsymbol{Do}\alpha \in SS\}, S_t)$ *satisfies IC.*

Given a set of numeric constraints, a "solution" is an assignment of values to the variables in those constraints that ensures that all the numeric constraints are satisfied. Feasible status sets are sets of ground status atoms which are assigned a 0-1 truth value

(those in the set are 1, those not in the set are 0) which satisfy a given agent program in a given state. Thus, the rules in the agent program and the state act as logical constraints that determine which status sets are feasible and which ones are not.

Example 4.2. *Consider the (initial) state presented in poss:sec:state, the red car agent program in ex:agent-programs, and the integrity and action constraints discussed in Sections 4.4.3 and 4.4.4, respectively. Let's focus on the red car. Suppose the red car has not received any request by other cars, and* conc *performs all actions in parallel determining the new positions of the red car given its speed, lane, etc.*

The status set SS consisting of the following status atoms is feasible:

$$\boldsymbol{P}accel(red, 2, 3, 1), \boldsymbol{P}continue(red, 1), \boldsymbol{P}decel(red, 2, 1, 1),$$
$$\boldsymbol{F}go\_left(red, 1), \boldsymbol{P}go\_right(red, 1), \boldsymbol{Do}continue(red, 1),$$
$$\boldsymbol{F}accel(red, S1, S2, 1) \text{ for every } S1 \text{ and every } S2 > 3,$$
$$\boldsymbol{F}decel(red, S1, S2, 1) \text{ for every } S1 \text{ and every } S2 < 1.$$

In fact, as per def:feasibleSS, the status set SS above satisfies

- *Conditions 1)–4), which can be easily verified;*
- *Condition 5), assuming that for each $\boldsymbol{P}\alpha$ in SS, the current state satisfies α's preconditions;*
- *Condition 6), as each status atom that should be derived from the agent program is indeed in SS;*
- *Condition 7), as all action constraints are satisfied by the $\boldsymbol{Do}\alpha$ status atoms in SS;*
- *Condition 8), as the new state satisfies the ICs.*

4.5. Pareto-optimal (Feasible) Status Sets

In any given state, an agent might have 0, 1, or several feasible status sets. Each feasible status (FSS) set SS has an associated set $\mathbf{Do}(SS) = \{\alpha \mid \mathbf{Do}\alpha \in SS\}$ of actions to be done if the agent chooses SS. Given an agent program, state, action and integrity constraints, FSSs are like solutions, just as sets of numeric constraints have solutions. Which FSS should an agent choose and act in accordance with?

In our SDC scenario, there can be different criteria a car might follow, e.g., a first criterion might minimize lane shifts (to increase safety); a second criterion might be to leave the highway at the exit closest to the destination. One feasible status set SS_1 might have it stay in the current lane, feasible status set SS_2 might make the car change lane on the right bringing it closer to the exit, while feasible status set SS_3 might make the car change lane on the left, making it further from the exit. Thus, SS_1 and SS_2 are incomparable in that SS_1 optimizes the first criterion but not the second, while the opposite holds for SS_2. On the other hand, SS_3 is strictly worse than both SS_1 and SS_2 and should be ruled out. Thus, an agent may use one or more criteria to select which of the several feasible status sets to base its actions on; such criteria are expressed via objective functions, defined below.

Definition 4.2. *An* objective function *objf is a mapping that assigns a real number to any given feasible status set SS. objf is said to be:*

(1) weakly monotonic *iff for any pair* SS_1, SS_2 *of feasible status sets,* $SS_1 \subseteq SS_2 \rightarrow$ *objf*$(SS_1) \leq$ *objf*(SS_2)*;*

(2) strongly monotonic *iff for any pair* SS_1, SS_2 *of feasible status sets,* $\{\alpha \mid \textbf{\textit{Do}}\alpha \in SS_1\} \subseteq \{\alpha \mid \textbf{\textit{Do}}\alpha \in SS_2\} \rightarrow \mathit{objf}(SS_1) \leq \mathit{objf}(SS_2)$*;*

(3) weakly anti-monotonic *iff for any pair* SS_1, SS_2 *of feasible status sets,* $SS_1 \subseteq SS_2 \rightarrow \mathit{objf}(SS_2) \leq \mathit{objf}(SS_1)$*;*

(4) strongly anti-monotonic *iff for any pair* SS_1, SS_2 *of feasible status sets,* $\{\alpha | \textbf{\textit{Do}}\alpha \in SS_1\} \subseteq \{\alpha \mid \textbf{\textit{Do}}\alpha \in SS_2\} \rightarrow \mathit{objf}(SS_2) \leq \mathit{objf}(SS_1)$.

In part (6) of the previous definition, the higher $\mathit{objf}(SS)$, the better SS is considered to be. As an example, an objective function that minimizes the number of lane shifts is defined as follows:

$$\mathit{objf}(SS) = -|\{\textbf{Do}\ \mathit{go\_left}(car, t) \in SS\} \cup \{\textbf{Do}\ \mathit{go\_right}(car', t') \in SS\}|.$$

We assume that each agent has an associated non-empty, finite set OF of objective functions. An agent will act in accordance with a feasible status set that is Pareto-optimal w.r.t. this set of functions.

Definition 4.3. *A feasible status set* $SS^\star$ *is* Pareto-optimal *w.r.t. a set* OF *of objective functions iff there is no other feasible status set* SS *such that for all* $\mathit{objf} \in OF$ $\mathit{objf}(SS) \geq \mathit{objf}(SS^\star)$ *and for some* $\mathit{objf} \in OF$ $\mathit{objf}(SS) > \mathit{objf}(SS^\star)$.

It is important to note that the above definition is key—it ties together the logical notion of a feasible status set (which is like a "solution" over a numeric domain) with the numeric notion of an objective function.

When only one objective function is present (i.e., $|OF| = 1$), Pareto-optimality coincides with the classical formulation of a (single objective function) optimization problem over the logical domain. That is, an optimal solution is a solution such that there is no other solution with a strictly better value for the objective function. In fact, with only one objective function *objf*, def:POSS states that a feasible status set $SS^\star$ is Pareto-optimal iff there is no other feasible status set SS such that $objf(SS) > objf(SS^\star)$.

In general, there could be zero, one, or many Pareto-optimal feasible status sets. In this case, we can choose one in several ways. One possibility is to choose any solution randomly—this is what is done in classical numerical optimization. However, additional options are also possible. We discuss these in Section 4.8.

We investigated the complexity of the central problem of deciding whether a given status set is a Pareto-optimal feasible status set. We start with the following proposition, which establishes an upper-bound under reasonable conditions.

Proposition 4.1. *If the agent program, the integrity constraints, the action constraints, and the action predicate names are fixed, and* conc *and the objective functions can be computed in polynomial time, then deciding whether a given status set SS is a Pareto-optimal feasible status set is in co-NP.*

Proof. We first show that deciding whether a status set SS' is feasible can be done in polynomial time under the assumptions in the statement. Conditions 1)–5) of def:feasibleSS can be clearly verified in polynomial time. Condition 6) can be verified in polynomial time because the agent program is fixed (and thus, there is a polynomial number of ground instances of operating rules). Condition 7) can be verified in polynomial

time because the action constrains are fixed. Condition 8) can be verified in polynomial time because *(i)* conc can be computed in polynomial time, *(ii)* checking constraint satisfaction can be done in polynomial time, since the integrity constrains are fixed.

We now show that the complementary problem, that is, deciding whether SS is *not* a Pareto-optimal feasible status set, is in NP. We first check whether SS is feasible; if not, then answer yes. As shown above this check can be done in polynomial time. If SS is feasible, then we guess a status set SS', and check that *(i)* SS' is feasible, and *(ii)* for all objective functions *objf*, $objf(SS') \geq objf(SS)$, and for some objective function *objf*, $objf(SS') > objf(SS)$. Check *(i)* can be done in polynomial time, as shown above. Check *(ii)* can be done in polynomial time because the objective functions can be computed in polynomial time. Also, SS' has polynomial size, since the actions' predicates are fixed. □

We now turn our attention to the lower-bound and show that deciding whether a given status set is a Pareto-optimal feasible one is co-NP-hard. In particular, co-NP-hardness holds even if the agent program, the integrity constraints, the action constraints (whose set is indeed empty), the action predicate names, and conc are fixed, there is only one fixed objective function, and conc and the objective functions can be computed in polynomial time.

Theorem 4.1. *Deciding whether a given status set is a Pareto-optimal feasible status set is co-NP-hard.*

Proof. We reduce the NP-hard 3-colorability problem to the complement of our problem, that is, deciding whether a status set SS is *not* a Pareto-optimal feasible status

set. An instance of 3-colorability is an undirected graph (V, E), for which it has to be decided whether there exists a *3-coloring*, that is, a way of assigning exactly one of three colors to every vertex in V so that no two adjacent (w.r.t. E) vertices have the same color. We derive an instance of the complement of our problem as follows. The initial state is $S_0 = \{\mathsf{vertex}(v) \mid v \in V\} \cup \{\mathsf{edge}(v, v') \mid (v, v') \in E\} \cup \{\mathsf{color}(c_1), \mathsf{color}(c_2), \mathsf{color}(c_3)\}$. The actions are as follows:

- For $v \in V$, we have action $\mathsf{dummycol}_a(v, c_1)$ with $Pre(\mathsf{dummycol}_a(v, c_1)) = true$, $Del(\mathsf{dummycol}_a(v, c_1)) = \emptyset$, and $Add(\mathsf{dummycol}_a(v, c_1)) = \{\mathsf{dummycol}(v, c_1), \mathsf{colored}(v)\}$.
- For $v \in V$, $c \in \{c_1, c_2, c_3\}$, action $\mathsf{coloring}_a(v, c)$ with $Pre(\mathsf{coloring}_a(v, c)) = true$, $Del(\mathsf{coloring}_a(v, c)) = \emptyset$, and $Add(\mathsf{coloring}_a(v, c)) = \{\mathsf{coloring}(v, c), \mathsf{colored}(v)\}$.
- For each $v \in V$, action $\mathsf{vertex}_a(v)$ with $Pre(\mathsf{vertex}_a(v)) = true$, $Del(\mathsf{vertex}_a(v)) = \emptyset$, and $Add(\mathsf{vertex}_a(v)) = \{\mathsf{vertex}_s(v)\}$.

The agent program contains $\mathbf{Do}\,\mathsf{vertex}_a(X) \leftarrow \mathsf{vertex}(X)$. The integrity constraints are:

$$\leftarrow \mathsf{coloring}(X, C_1)\,\&\,\mathsf{dummycol}(Y, C_2)$$
$$\leftarrow \mathsf{edge}(X, Y)\,\&\,\mathsf{coloring}(X, C)\,\&\,\mathsf{coloring}(Y, C)$$
$$\leftarrow \mathsf{coloring}(X, c_1)\,\&\,\mathsf{coloring}(X, c_2)$$
$$\leftarrow \mathsf{coloring}(X, c_1)\,\&\,\mathsf{coloring}(X, c_3)$$
$$\leftarrow \mathsf{coloring}(X, c_2)\,\&\,\mathsf{coloring}(X, c_3)$$
$$\mathsf{colored}(X) \leftarrow \mathsf{vertex}_s(X)$$

The set of action constraints is empty. We also have $\mathsf{conc}(A, S_t) = S_t \setminus (\bigcup_{\alpha \in A} Del(\alpha)) \cup \bigcup_{\alpha \in A} Add(\alpha)$ and $objf(SS) = |\{\mathbf{Do}\,\mathsf{coloring}_a(v, c) \in SS\}|$. The status set SS contains $\mathbf{Do}\,\mathsf{vertex}_a(v)$, $\mathbf{P}\,\mathsf{vertex}_a(v)$, $\mathbf{Do}\,\mathsf{dummycol}_a(v, c_1)$, $\mathbf{P}\,\mathsf{dummycol}_a(v, c_1)$, for each $v \in V$. We

now show that (V, E) has a 3-coloring iff SS is *not* a Pareto-optimal feasible status set. First of all, we point out that SS is feasible and $objf(SS) = 0$, which can be easily verified.

($\Rightarrow$) Let $\phi : V \to \{c_1, c_2, c_3\}$ be a 3-coloring of (V, E). We first show that the following status set is feasible:

$$\begin{aligned} SS' = \; & \textstyle\bigcup_{v \in V} \{\mathbf{Do}\, \mathsf{vertex}_a(v), \mathbf{P}\, \mathsf{vertex}_a(v)\} \cup \\ & \textstyle\bigcup_{v \in V} \{\mathbf{Do}\, \mathsf{coloring}_a(v, \phi(v)), \mathbf{P}\, \mathsf{coloring}_a(v, \phi(v))\} \end{aligned}$$

Conditions 1)–4) of def:feasibleSS are clearly satisfied by SS'. Condition 5) is satisfied, as all action preconditions are trivially true. Condition 6) is satisfied since for each $\mathsf{vertex}(v)$ in S_t, $\mathbf{Do}\, \mathsf{vertex}_a(v)$ is included in SS'. Condition 7) is satisfied because there are no action constraints. Let us now discuss Condition 8). Notice that S_0 satisfies the ICs. We need to show that $S_1 = \mathsf{conc}(\{\alpha \mid \mathbf{Do}\alpha \in SS'\}, S_t)$ satisfies the ICs. By definition of conc, and the actions' *Del* and *Add* sets, $S_1 = S_0 \cup \{\mathsf{vertex}_s(v) \mid v \in V\} \cup \bigcup_{v \in V} \{\mathsf{coloring}(v, \phi(v)), \mathsf{colored}(v)\}$. Since ϕ is a 3-coloring, it can be easily verified that all ICs are satisfied by S_1. Hence, SS' is a feasible status set and $objf(SS') = |V|$. W.l.o.g. we can assume the original graph has at least one vertex and thus $objf(SS') > 1$, and thus SS is not Pareto-optimal.

($\Leftarrow$) Suppose (V, E) has no 3-coloring. We show that there is no feasible status set SS' containing at least one status atom of the form $\mathbf{Do}\, \mathsf{coloring}_a(v, c)$—which implies that SS is Pareto-optimal. Reasoning by contradiction, suppose SS' exists. In order for SS' to be feasible, it must satisfy Condition 6) of def:feasibleSS, and thus SS' must include $\{\mathbf{Do}\, \mathsf{vertex}_a(v) \mid v \in V\}$. This means that the new state S_1 will include $\{\mathsf{vertex}_s(v) \mid v \in V\}$, as per definition of conc and the *Add* sets for $\mathsf{vertex}_a(v)$ actions. In order for

S_1 to satisfy the last IC, S_1 must include $\{\mathsf{colored}(v) \mid v \in V\}$. Since SS' includes at least one status atom of the form $\mathbf{Do}\,\mathsf{coloring}_a(v, c)$, S_1 includes $\mathsf{coloring}(v, c)$, and thus SS' cannot include any status atom of the form $\mathbf{Do}\,\mathsf{dummycol}_a(v', c_1)$, because otherwise $\mathsf{dummycol}(v', c_1)$ would be in S_1 violating the first IC. Thus, the only way for S_1 to have an atom $\mathsf{colored}(v)$ for each vertex $v \in V$ is that SS' has at least one $\mathbf{Do}\,\mathsf{coloring}_a(v, c)$ status atom for each vertex $v \in V$. Notice that each status atom $\mathbf{Do}\,\mathsf{coloring}_a(v, c)$ yields the atom $\mathsf{coloring}(v, c)$ in S_1. In order for S_1 to satisfy the third to fifth ICs, S_1 must contain at most one $\mathsf{coloring}(v, c)$ atom for each vertex v. Thus, S_1 contains exactly one $\mathsf{coloring}(v, c)$ atom for each vertex v. Notice that S_1 must satisfy also the second IC. Now it is easy to see that the function assigning to each vertex v the color c iff $\mathsf{coloring}(v, c)$ belongs to S_1 is a 3-coloring, which is a contradiction. □

From the results above, we get the following corollary.

Corollary 4.1. *If the agent program, the integrity constraints, the action constraints, and the action predicate names are fixed, and* conc *and the objective functions can be computed in polynomial time, then deciding whether a given status set is a Pareto-optimal feasible status set is co-NP-complete.*

4.6. Algorithms

In this section, we introduce several algorithms to compute Pareto-optimal feasible status sets.

First, we present a "helper" algorithm (used by all other algorithms) to compute the "closure" of a status set (poss:alg:closure). Then, we propose a baseline algorithm that can be used with arbitrary sets of objective functions (poss:alg:poss-naive). Next, we develop

exact algorithms for weakly/strongly anti-monotonic objective functions (Algorithms 3–4). These methods leverage anti-monotonicity to improve on the baseline. Their basic idea is to traverse up a lattice of status sets in a breadth-first fashion, where the lattice is defined w.r.t. set-inclusion (resp., set-inclusion of $\mathbf{Do}\alpha$ atoms) for weakly (resp., strongly) anti-monotonic objective functions. This strategy allows the algorithms to start from the "smallest" possibly feasible status sets, look for a Pareto-optimal feasible one, and move to bigger status sets only if needed.

A similar idea can be applied to weakly and strongly monotonic objective functions, but the lattice is traversed downwards starting from the "biggest" possibly feasible status sets. We found this strategy less effective compared to the anti-monotonic case, because the biggest status sets to start from may contain many contradictory status atoms (e.g., violating action constraints) and moving to smaller feasible ones might require traversing several levels of the lattice. For this reason, with weakly/strongly monotonic objective functions, in order to significantly improve on the baseline algorithm, we introduced heuristics leading to the two approximation algorithms presented in the following (poss:alg:poss-sma,poss:alg:poss-wma).

All algorithms in this section except for the "helper" one take as input: a state S_t, an agent program P, a set IC of integrity constraints, a set AC of action constraints, a conc function, a set OF of objective functions, and a set A of ground actions. poss:alg:poss-sma,poss:alg:poss-wma have an additional input τ, which is used for the heuristic search and will be discussed later.

4.6.1. Helper Algorithm

The Closure algorithm (cf. poss:alg:closure) takes as input a status set SS, a current state S_t, an agent program P, and a set DC of denial action constraints. The goal of the algorithm is to compute a status set that includes SS and satisfies Conditions 1)-6) of def:feasibleSS, as well as Condition 7) w.r.t. denial action constraints only, if such a status set exists. If a status set is returned, it might not be feasible, as Condition 7) of def:feasibleSS w.r.t. definite action constraints, as well as the last condition of def:feasibleSS, still need to be verified.

The algorithm first "closes" SS w.r.t. Conditions 1)–3) of def:feasibleSS (lines 1–6). It then checks if Conditions 4), 5), and 7) are all satisfied (lines 7–10). If any of them is not satisfied, then $\perp$ is returned. Otherwise, the algorithm iteratively enforces Condition 6) of def:feasibleSS (lines 11–26), thereby possibly deriving further ground status atoms. While doing so, the algorithms enforces Conditions 1)–3) of def:feasibleSS (lines 18–21) and checks that Conditions 4)–5) and Condition 7) (w.r.t. the denial action constraints in DC) of def:feasibleSS remain satisfied w.r.t. the ground status atoms that are being derived (lines 22–25)—once again, if any condition is violated, $\perp$ is returned, otherwise the algorithms keeps adding new ground status atoms until a fixpoint is reached and the resulting set is returned (line 27).

It is worth noting that every ground status atom derived by the algorithm must be in any status set SS' extending SS in order for SS' to be possibly feasible. A status set returned by the algorithm that satisfies also Condition 7) of def:feasibleSS w.r.t. all action constraints as well as Condition 8) is feasible.

Algorithm 1 Closure

Input: A status set SS, a state S_t, an agent program P, and
a set DC of denial action constraints.
Output: A status set or $\perp$.
for each $\mathbf{O}\alpha \in SS$ s.t. $\mathbf{P}\alpha \notin SS$ **do**
  Add $\mathbf{P}\alpha$ to SS.
end for
for each $\mathbf{O}\alpha \in SS$ s.t. $\mathbf{Do}\alpha \notin SS$ **do**
  Add $\mathbf{Do}\alpha$ to SS.
end for
for each $\mathbf{Do}\alpha \in SS$ s.t. $\mathbf{P}\alpha \notin SS$ **do**
  Add $\mathbf{P}\alpha$ to SS.
end for
if there exists α s.t. (i) $\{\mathbf{P}\alpha, \mathbf{F}\alpha\} \subseteq SS$ or (ii) $\mathbf{P}\alpha \in SS$ and $Pre(\alpha)$ is false in S_t **then**
  return $\perp$.
end if
if $\{\alpha \mid \mathbf{Do}\alpha \in SS\}$ does not satisfy DC **then**
  return $\perp$.
end if
$SS' := SS$.
repeat
  $SS'' := SS'$.
  for each ground rule r of P **do**
    Let r be $SA \leftarrow \chi$ & SA_1 & ... & SA_n.
    if χ is true in S_t and $\{SA_1, \ldots, SA_n\} \subseteq SS'$ **then**
      Add SA to SS'.
      if $SA = \mathbf{O}\alpha$ **then**
        Add $\mathbf{P}\alpha$ and $\mathbf{Do}\alpha$ to SS'.
      else if $SA = \mathbf{Do}\alpha$ **then**
        Add $\mathbf{P}\alpha$ to SS'.
      end if
      if there exists α s.t. (i) $\{\mathbf{P}\alpha, \mathbf{F}\alpha\} \subseteq SS'$ or (ii) $\mathbf{P}\alpha \in SS'$ and $Pre(\alpha)$ is false in S_t **then**
        return $\perp$.
      end if
      if $\{\alpha \mid \mathbf{Do}\alpha \in SS'\}$ does not satisfy DC **then**
        return $\perp$.
      end if
    end if
  end for
until $SS' = SS''$
return SS'.

The proposition below states an important property that will be leveraged by the algorithms introduced in the following.

Proposition 4.2. *Let* $LSS = \mathsf{Closure}(\emptyset, S_t, P, DC)$ *for any status* S_t*, agent program* P*, and set of denial action constraints* DC*. If* $LSS = \bot$*, then there is no feasible status set. If* $LSS \neq \bot$*, every feasible status set (if any) contains* LSS*.*

Proof. When Closure is called with $SS = \emptyset$, lines 1–10 have no effect. Then, lines 11-27 are executed, enforcing Conditions 1)–3) and 6) of def:feasibleSS by possibly deriving new status atoms. Such status atoms must be necessarily contained in any feasible status set containing the empty set, and thus in every feasible status set (if any). Recall that lines 11-27 additionally check whether any of Conditions 4), 5), and 7) of def:feasibleSS is violated. If a status set violates any of such conditions, then every superset of it violates the same conditions. Thus, Closure returns $\bot$ when the set SS' of status atoms currently computed (which must be included in every feasible status set, if any) violates any of Conditions 4), 5), and 7) (which will be violated by every superset of SS'), that is, there is no feasible status set. If Closure returns a status set, the latter does not violate any of Conditions 4), 5), and 7) and must be contained in every feasible status set, if any. □

In the sequel, we use the following notation. For any program P, we use g_P (resp., χ_P, b_P) to denote the number of ground rules of P (resp., the maximum number of atoms in the condition χ of rules in P, the maximum number of status atoms of rules in P). For any set of constraints C, we use $||C||$ to denote the overall number of atoms in C. As customary, for any set X, we use $|X|$ to denote the cardinality of X. Finally, we use A to denote the set of all ground actions.

Proposition 4.3. *The worst-case time complexity of poss:alg:closure is* $O(|A| \cdot g_P \cdot (\chi_P \cdot |S_t| + |A| \cdot (b_P + lg|A| + |S_t| + ||DC||)))$.

4.6.2. Baseline Algorithm

We now introduce a baseline algorithm (POSS baseline, cf. poss:alg:poss-naive) to compute a Pareto-optimal feasible status set (if one exists) with an arbitrary set of objective functions.

Given a set A of actions, we define $SA(A) = \{Op\,\alpha \mid \alpha \in A \text{ and } Op \in \{\mathbf{F}, \mathbf{P}, \mathbf{O}, \mathbf{Do}\}\}$.

Algorithm 2 POSS baseline

Input: A state S_t, an agent program P,
a set IC of integrity constraints,
a set AC of action constraints, a conc function,
a set OF of objective functions, and
a set A of ground actions.
Output: A Pareto-optimal feasible status set or $\perp$.
Let DC be the set of denial constraints in AC.
$LSS = \mathsf{Closure}(\emptyset, S_t, P, DC)$.
if $LSS = \perp$ **then**
  return $\perp$.
end if
$\overline{A} := \{\alpha \mid \alpha \in A$ and ($Pre(\alpha)$ *is false in* S_t or $\mathbf{F}\alpha \in LSS$)$\}$.
$\overline{SA} := \cup_{\alpha \in \overline{A}} \{\mathbf{Do}\alpha, \mathbf{O}\alpha, \mathbf{P}\alpha\}$.
$SA := SA(A) \setminus \overline{SA}$.
$\mathcal{S} = \emptyset$.
for each SS s.t. $LSS \subseteq SS \subseteq SA$ **do**
  if SS is a feasible status set **then**
    Add SS to $\mathcal{S}$.
  end if
end for
if $\mathcal{S} = \emptyset$ **then**
  return $\perp$.
else
  return a Pareto-optimal (w.r.t. OF) element of $\mathcal{S}$.
end if

The algorithm first calls the Closure algorithm with the empty status set, the current state, the agent program, and the denial action constraints in AC, thereby getting LSS (lines 1–2). If LSS is $\bot$, then there is no feasible status set and the algorithm returns $\bot$ (lines 3–4). Otherwise, there might exist feasible status sets, and if any exists it has to contain LSS. For this reason, lines 1–4 will be replicated in all our algorithms reported in the following. Thus, the algorithm looks for feasible status sets that are a superset of LSS (lines 8–11), and if none exists $\bot$ is returned (lines 12-13), otherwise a Pareto-optimal one is returned (lines 14–15). Moreover, a simple pruning is applied when searching for feasible status sets containing LSS. The algorithm ignores status atoms that cannot be in any feasible status set (lines 5–7): these are the $\mathbf{Do}\alpha$, $\mathbf{O}\alpha$, and $\mathbf{P}\alpha$ status atoms for which $Pre(\alpha)$ is false in the current state (see Conditions 1)–3) and 5) of def:feasibleSS) or $\mathbf{F}\alpha$ belongs to LSS (see Conditions 1)–4) of def:feasibleSS). Such a pruning will be applied by all algorithms presented in the following as well.

Theorem 4.2. *poss:alg:poss-naive correctly computes a Pareto-optimal feasible status set.*

Proof. By pro:LSS, if $LSS = \bot$ in line 3, then there is no feasible status set and the algorithm correctly returns $\bot$. Otherwise, by pro:LSS, LSS is a status set that must be contained in every feasible status set, if one exists. The algorithm looks for feasible status sets SS s.t. $LSS \subseteq SS \subseteq SA$, and returns a Pareto-optimal one among them, if at least one feasible status set has been found. So, to prove correctness, we need to show that no feasible status set is missed by the algorithm, that is, there is no feasible status set SS s.t. $SS \subsetneq LSS$ or $SS \supsetneq SA$. pro:LSS implies that there cannot be any feasible status set SS

s.t. $SS \subsetneq LSS$. Notice that each status atom $\mathbf{P}\alpha$ s.t. $Pre(\alpha)$ is false in S_t or $\mathbf{F}\alpha \in LSS$ cannot be included in any feasible status set. For such $\mathbf{P}\alpha$ status atoms, the status atoms $\mathbf{O}\alpha$ and $\mathbf{Do}\alpha$ cannot be included in any feasible status set too, because of Conditions 1) and 3) of def:feasibleSS. Thus, lines 5–7 safely disregard the status atoms in $\overline{SA}$, as they cannot belong to any feasible status set, and hence there cannot be a feasible status set $SS \supsetneq SA$. □

Proposition 4.4. *The worst-case time complexity of poss:alg:poss-naive is* $O(|A|^2 \cdot g_P \cdot ||DC|| + 2^{2|A|} \cdot f_{OF}(A) + 2^{|A|} \cdot (|A| \cdot lg|A| + |A| \cdot |S_t| + g_P \cdot (|S_t| \cdot \chi_P + |A| \cdot b_P) + |A| \cdot ||AC|| + |S_t| \cdot ||IC|| + f_{\mathsf{conc}}(|A|, |S_t|)))$*, where* f_{OF} *(resp.,* f_{conc}*) is the function measuring the worst-case time complexity of evaluating the objective functions in* OF *(resp.,* conc*).*

The numbers of cars and lanes affect number of rules in the program and the size of the constraints (g_P, $||IC||$, $||DC||$) as well as the number of actions ($|A|$). Such observations apply also to the other algorithms presented in the following.

4.6.3. Weakly and Strongly Anti-Monotonic Algorithms

We propose algorithms to compute Pareto-optimal feasible status sets in the presence of weakly (cf. poss:alg:poss-wam) and strongly (cf. poss:alg:poss-sam) *anti-monotonic* objective functions.

Let us start with poss:alg:poss-wam. The basic idea of the algorithm is to traverse a lattice (w.r.t. set-inclusion) of status sets where the bottom element is the set LSS computed in lines 1–2. In particular, the lattice is traversed upwards starting from LSS in a breadth-first fashion. In lines 1–6, the algorithm applies the same pruning discussed

Algorithm 3 POSS weakly-anti-monotonic

Input: A state S_t, an agent program P,
1: a set IC of integrity constraints,
2: a set AC of action constraints, a conc function,
3: a set OF of weakly anti-monotonic objective functions, and
4: a set A of ground actions.
Output: A Pareto-optimal feasible status set or $\perp$.
5: Let DC be the set of denial constraints in AC.
6: $LSS = \mathsf{Closure}(\emptyset, S_t, P, DC)$.
7: **if** $LSS = \perp$ **then**
8: **return** $\perp$.
9: **end if**
10: $\overline{A} := \{\alpha \mid \alpha \in A$ and $(Pre(\alpha)$ *is false in* S_t or $\mathbf{F}\alpha \in LSS)\}$.
11: $\overline{SA} := \cup_{\alpha \in \overline{A}} \{\mathbf{Do}\alpha, \mathbf{O}\alpha, \mathbf{P}\alpha\}$.
12: $SA := SA(A) \setminus (\overline{SA} \cup LSS)$.
13: $ToInspect := \{LSS\}$.
14: **while** $ToInspect \neq \emptyset$ **do**
15: $Candidates := ToInspect$.
16: $ToInspect := \emptyset$.
17: **if** *Candidates* has a feasible status set **then**
18: **return** a Pareto-optimal (w.r.t. OF) feasible status set of *Candidates*.
19: **else**
20: **for each** $Cand$ in *Candidates* **do**
21: **for each** $Op\,\alpha \in (SA \setminus Cand)$ **do**
22: **if** $(Cand \cup \{Op\,\alpha\}) \notin ToInspect$ **then**
23: Add $Cand \cup \{Op\,\alpha\}$ to *ToInspect*.
24: **end if**
25: **end for**
26: **end for**
27: **end if**
28: **end while**
29: **return** $\perp$.

before for the baseline algorithm. Then, SA consists of the status atoms that might be added to LSS (line 7). In lines 8–18, the algorithm performs the aforementioned traversal of the lattice, one level at a time, starting from LSS, where each level is built by adding one status atom to each status set of the previous level (see lines 15–18). When a feasible status set exists in a level, a Pareto-optimal one is returned, otherwise the next level is considered. It is worth noting that each level is built only if needed and the lattice is not

entirely materialized at once, which yields computational benefits in terms of both run time and memory usage. Eventually, if no feasible status set has been encountered, $\perp$ is returned (line 19).

Theorem 4.3. *poss:alg:poss-wam correctly computes a Pareto-optimal feasible status set.*

Proof. The same argument in the proof of th:baseline-correctness applies to lines 1–6 of poss:alg:poss-wam. Thus, the status atoms in $LSS \cup SA$ are the only ones that can possibly belong to a feasible status set. It is easy to see that (in lines 8–19) the algorithm starts from LSS and then iteratively considers bigger status sets, where at each iteration (of the **while** loop in lines 9–18) status sets that are incomparable w.r.t. set-inclusion are considered. At a generic iteration, if a feasible status is found that is Pareto-optimal among those considered in that iteration, then it must be Pareto-optimal also w.r.t. bigger status sets, because objective functions are weakly anti-monotonic. □

poss:alg:poss-sam deals with strongly anti-monotonic objective functions, and behaves like poss:alg:poss-wam, except that the lattice is built w.r.t. set-inclusion of **Do**α status atoms.

Theorem 4.4. *poss:alg:poss-sam correctly computes a Pareto-optimal feasible status set.*

Proof. The same argument in the proof of th:poss-wam-correctness applies, noting that status sets are compared w.r.t. **Do**α status atoms, because objective functions are strongly monotonic. □

Algorithm 4 POSS strongly-anti-monotonic

Input: A state S_t, an agent program P,
a set IC of integrity constraints,
a set AC of action constraints, a conc function,
a set OF of strongly anti-monotonic objective functions, and
a set A of ground actions.
Output: A Pareto-optimal feasible status set or $\bot$.
Let DC be the set of denial constraints in AC.
$LSS = \mathsf{Closure}(\emptyset, S_t, P, DC)$.
if $LSS = \bot$ **then**
  return $\bot$.
end if
$\overline{A} := \{\alpha \mid \alpha \in A$ and $(Pre(\alpha)$ *is false in* S_t or $\mathbf{F}\alpha \in LSS)\}$.
$\overline{SA} := \cup_{\alpha \in \overline{A}} \{\mathbf{Do}\alpha, \mathbf{O}\alpha, \mathbf{P}\alpha\}$.
$SA := SA(A) \setminus (\overline{SA} \cup LSS)$.
$SA\text{-}Do := \{\mathbf{Do}\alpha \mid \mathbf{Do}\alpha \in SA\}$.
$SA\text{-}FPO := SA \setminus SA\text{-}Do$.
$ToInspect := \{LSS \cup X \mid X \subseteq SA\text{-}FPO\}$.
while $ToInspect \neq \emptyset$ **do**
  $Candidates := ToInspect$.
  $ToInspect := \emptyset$.
  if *Candidates* has a feasible status set **then**
    return a Pareto-optimal (w.r.t. OF) feasible status set of *Candidates*.
  else
    for each *Cand* in *Candidates* **do**
      for each $\mathbf{Do}\alpha \in (SA\text{-}Do \setminus Cand)$ **do**
        if $(Cand \cup \{\mathbf{Do}\alpha\}) \notin ToInspect$ **then**
          Add $Cand \cup \{\mathbf{Do}\alpha\}$ to *ToInspect*.
        end if
      end for
    end for
  end if
end while
return $\bot$.

The worst-case time complexity of poss:alg:poss-wam,poss:alg:poss-sam is the one stated in th:baseline-complexity, as in the worst case, $O(2^{|A|})$ candidate status sets still need to be inspected. While this is a theoretical analysis in the *worst case*, we will show in poss:sec:experiments that poss:alg:poss-wam,poss:alg:poss-sam indeed provide computational benefits over the baseline in practice.

4.6.4. Weakly and Strongly Monotonic Algorithms

In this section, we introduce approximation algorithms for weakly and strongly *monotonic* objective functions.

Let us start with poss:alg:poss-wma, which deals with weakly-monotonic objective functions. The basic idea is to start with the biggest "possibly feasible" status sets, and then move to smaller ones if needed (i.e., if no feasible status set has been found). Lines 1–5 are analogous to the ones of the algorithms discussed so far. In lines 6–14, the algorithm builds the biggest possibly feasible status sets to start from, applying different pruning strategies that rule out status sets that are not feasible for sure. First, the algorithm rules out status atoms of the form $\mathbf{O}\alpha$, $\mathbf{Do}\alpha$, and $\mathbf{P}\alpha$ for which $Pre(\alpha)$ is not satisfied in the current state or $\mathbf{F}\alpha$ belongs to *LSS*—moreover, for such actions α, all $\mathbf{F}\alpha$ status atoms are included, as they will not conflict for sure with any $\mathbf{O}\alpha$, $\mathbf{Do}\alpha$, or $\mathbf{P}\alpha$ status atom (line 6). Second, for actions α not satisfying the aforementioned conditions, to construct the biggest status sets, either $\{\mathbf{F}\alpha\}$ or $\{\mathbf{O}\alpha, \mathbf{Do}\alpha, \mathbf{P}\alpha\}$ is considered (lines 7–14) to avoid status sets that would not be feasible. The **while** loop in lines 16–31 starts from the biggest status sets and moves to smaller ones if no feasible one has been found. At each iteration, only τ (randomly picked) status sets from *ToInspect* are considered (see lines 17–18), where τ is an additional input of the algorithm. The status sets in *ToInspect* that are not chosen by the random sampling are still left in *ToInspect* for later inspection. This allows the algorithm to move faster to lower levels of the status set lattice, which pays off in terms of running time, as we show in our experimental evaluation. Of course, the algorithm might return sub-optimal feasible status sets, because when a set of feasible

status sets is considered and a Pareto-optimal one is determined among them (lines 19–20), some other better feasible status sets might have been ignored (not being chosen by the random sampling).

Smaller status sets are built from the current ones by deleting a single status atom (lines 23–30), following the following criteria. A status atom of the form $\mathbf{Do}\alpha$ is deleted from a status set if the latter does not contain $\mathbf{O}\alpha$ (lines 27–28), because otherwise the deletion would yield a non-feasible status set—see Condition 2) of def:feasibleSS. Likewise, a status atom of the form $\mathbf{P}\alpha$ is deleted from a status set if the latter contains neither $\mathbf{O}\alpha$ nor $\mathbf{Do}\alpha$ (lines 29–30), because otherwise the deletion would yield a non-feasible status set—see Conditions 1) and 3) of def:feasibleSS. A status atom of the form $\mathbf{O}\alpha$ or $\mathbf{F}\alpha$ is deleted without checking further conditions (lines 25–26), as their deletion does not yield violations of Conditions 1)–4) of def:feasibleSS. The algorithm returns $\perp$ if no feasible status set is eventually found (line 32).

Let us consider now poss:alg:poss-sma. The algorithm starts from "possibly feasible" status sets containing as many $\mathbf{Do}\alpha$ status atoms as possible, which are collected into *ToInspect* (lines 1–9). The algorithm leverages the following ideas discussed before: it moves to smaller status sets if no feasible one has been found; it applies the sampling approach of poss:alg:poss-wma; it reduces the number of status sets to be considered when initializing *ToInspect* (in lines 7–9) and when a lower level of the lattice has to be built (see lines 18–25).

The worst-case time complexity of poss:alg:poss-wma,poss:alg:poss-sma is the one stated in th:baseline-complexity, as in the worst case, $O(2^{|A|})$ candidate status sets still need to be inspected. While this is a theoretical analysis in the *worst case*, poss:sec:experiments

will show that poss:alg:poss-wma,poss:alg:poss-sma provide computational benefits over the baseline in practice.

4.7. Experimental Assessment

We varied the parameters reported in Table 4.1, where the default value we fixed for a parameter when varying another parameter is highlighted in bold.

Table 4.1. Varying parameters (default values in bold).

Parameter	*Values*
Number of red cars	$\{1, \mathbf{10}, 20, 30\}$
Number of orange cars	$\{1, \mathbf{10}, 20, 30\}$
Number of green cars	$\{1, \mathbf{10}, 20, 30\}$
Number of lanes	$\{\mathbf{6}, 8, 10\}$ (plus exit lane)
Highway length	$\{40, 60, \mathbf{80}, 100\}$

In addition:

- We randomly picked the initial position of each car, ensuring that (i) two cars are not in the same cell and (ii) all initial positions are before the 10th cell.
- We fixed the speed of each car as follows:
  - For red cars, randomly picking from $\{1, 2, 3\}$.
  - For green cars: randomly picking from $\{1, 2\}$.
  - For orange cars: equal to 1.
- We randomly picked the destination of each car from $\{A, B, C, D, E\}$.
- We fixed the number of exits to a tenth of the highway length, with the first exit positioned at the 10th cell and with a distance of 10 cells between two consecutive exits.

- We randomly picked the destinations associated with each exit from $\{A, B, C, D, E\}$, ensuring that each destination has at least one associated exit.
- For the approximation algorithms, τ was set to 10 (early experiments showed this to be a good value).
- We used three objective functions: *Lane Shift Penalty* (LSP), which penalizes lane shifts in a status set, *Exit Miss Penalty* (EMP), which penalizes a status set that makes cars miss the exit (in two time steps), and *Change Speed Penalty* (CSP), which penalizes a status set that does not speed up when the exit is too far or does not slow down when the exit is very close. All objective functions are weakly and strongly anti-monotonic; as weakly and strongly monotonic objective functions, we used the same ones but with a flipped sign.

All experiments were performed on a machine with 36 Intel Core i9-10980XE CPUs, 256GB RAM, running Ubuntu 18.04.

4.7.1. Runtime

Figure 4.3 reports the average time needed to compute a POSS when varying the number of cars of each color, the number of lanes, and the highway length. In all the figures, (i) POSS Baseline (M) and POSS Baseline (A-M) correspond to Baseline using monotonic and anti-monotonic objective functions, respectively, (ii) orange lines represent algorithms using monotonic objective functions, and (iii) blue lines represent algorithms using anti-monotonic objective functions.

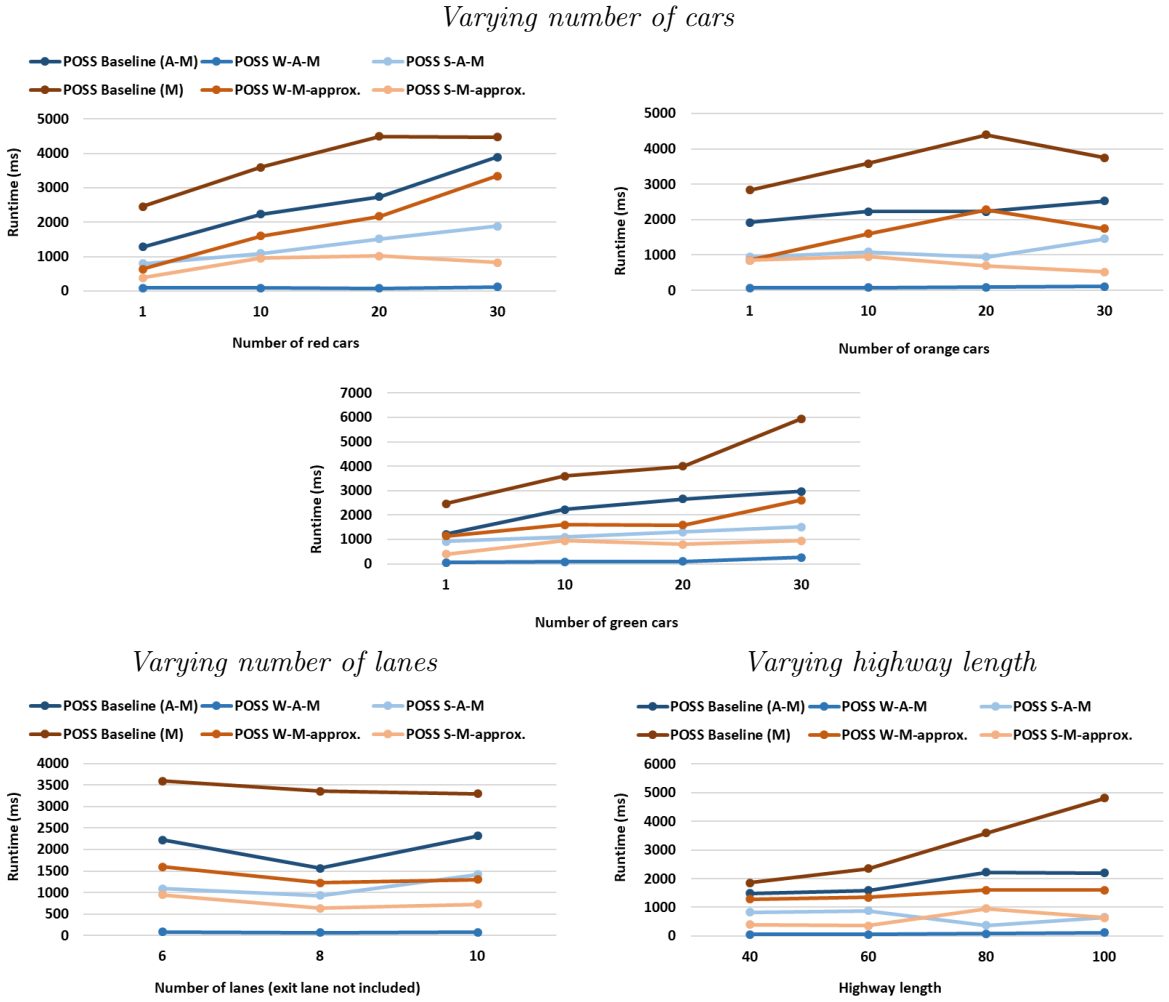


Figure 4.3. Runtimes obtained when varying: (top) number of cars of each color, (bottom left) number of lanes, and (bottom right) highway length.

The results show that the runtime of POSS W-A-M is the best, followed by POSS S-M-approx. As POSS W-A-M is an exact algorithm, this suggests that we should try to convert true objective functions into similar weakly anti-monotonic ones.

In all, we compared our 4 proposed algorithms with Baseline (i) in 48 cases when varying the number of cars of each color (4 algorithms $\times$ 4 parameter values $\times$ 3 colors), (ii) in 12 cases when varying the number of lanes (4 algorithms $\times$ 3 parameter values), and

(iii) in 16 cases when varying the highway length (4 algorithms × 4 parameter values). The results show that (i) our proposed algorithms are faster than Baseline in all of the 76 cases and (ii) on average, our proposed algorithms run much faster than Baseline—see Table 4.2, where the performance gain of each algorithm Alg is computed as $1-\frac{time(\text{Alg})}{time(\text{Baseline})}$.

Table 4.2. Average performance gain vs. Baseline.

Varying parameter	POSS W-A-M	POSS S-A-M	POSS W-M-approx.	POSS S-M-approx.
# red cars	95.98%	46.31%	51.58%	79.18%
# orange cars	96.02%	50.60%	56.78%	78.50%
# green cars	94.76%	43.90%	56.47%	80.28%
# lanes	96.17%	43.51%	59.77%	77.52%
Highw. len.	95.98%	60.61%	48.86%	80.90%

4.7.2. Solution Quality for Approximate Algorithms

We also assessed the quality of the solutions computed by our approximate algorithms POSS W-M-approx. and POSS S-M-approx. Inverted generational distance (IGD) and hypervolume (HV) are two popular approaches for measuring the solution quality of approximate algorithms for multi-objective problems (29). IGD measures the distance between the objective values obtained by the approximate algorithm and the values in the Pareto front (i.e., the set of objective values corresponding to a set of Pareto-optimal feasible status sets), and HV measures the diversity and convergence by calculating the volume between the objective values obtained from the approximate algorithms and specified reference points. Here, our objective functions are weakly or strongly anti-monotonic/monotonic, and the optimal objective values do not vary much. Therefore, we use the IGD approach to measure the quality of the solutions computed by our approximate algorithms. Instead of simply calculating the distance, we looked at the relative

quality $\frac{value(objf,\mathsf{Alg})}{value(objf,\mathsf{Baseline})}$, where $value(objf, \mathsf{Alg})$ is the value obtained for objective function *objf* using Alg. Table 4.3 reports the average values of the objective functions introduced above, when varying the various parameters. The results show that both POSS W-M-approx. and POSS S-M-approx. are able to provide good quality solutions. The average relative quality using POSS W-M-approx. ranged from 62.9% to 66.5% for LSP, from 61.8% to 74.3% for EMP, and from 38.2% to 77.4% for CSP. POSS S-M-approx. provided even better results—its average relative quality ranged from 83.6% to 98.4% for LSP, from 93.1% to 99.0% for EMP, and from 76.5% to 98.9% for CSP. Compared to a fully random approximation algorithm, the overall relative quality provided by POSS W-M-approx. and POSS S-M-approx. was much higher (on average, 34.6% for LSP, 32.9% for EMP, and 42.6% for CSP).

Table 4.3. Average relative quality vs. Baseline. LSP is Lane Shift Penalty, EMP is Exit Miss Penalty, and CSP is Change Speed Penalty.

Varying parameter	POSS W-M-approx.			POSS S-M-approx.		
	LSP	EMP	CSP	LSP	EMP	CSP
# red cars	64.0%	67.9%	77.4%	95.0%	98.1%	92.1%
# orange cars	66.2%	70.5%	75.3%	95.9%	99.0%	97.9%
# green cars	63.0%	61.8%	76.6%	83.6%	93.1%	76.5%
# lanes	62.9%	74.3%	60.0%	96.3%	97.5%	98.9%
Highw. len.	66.5%	64.4%	38.2%	98.4%	99.0%	88.5%

4.8. Choosing an Optimal Feasible Status Set

There can be situations where the Pareto frontier contains many Pareto-optimal status sets $PF = \{SS_1, \ldots, SS_n\}$. When this happens, the agent in question must choose one of these status sets even though all of these are deemed optimal according to the set of

objective functions that were explicitly stated. In this case, many solutions are possible. We briefly discuss these below.

Random Choice. One possibility is for the agent to randomly choose one of the SS_j's and take the actions articulated therein. This method is fast and may be appropriate in cases where the agent needs to act very quickly and a near real-time choice must be made.

Weighted Objective Functions. (43; 120) have suggested that a Pareto-optimal solution be chosen according to weights that the system designer associates with each objective function. In this case, we associate a score with each SS_j in the Pareto frontier which is set to a linear combination (using the weights) of each objective function value. The SS_j with the highest score is then chosen.

Clustering-based Approaches. The clustering-based approach (120; 77; 78; 24) is also an important approach for selecting Pareto-optimal solutions. (77; 78) develop theories and procedures for selecting and clustering multiple criteria solutions. They proposed that the mutually exclusive clusters are determined by (i) the similarities between the solutions, and (ii) the decision-maker's preference structure. The procedures for making a decision include (i) generating optimal solutions, (ii) clustering solutions based on their similarities, and (iii) selecting one or more solutions from each cluster. Specifically, (77) used artificial neural networks (ANN) with variable weights for clustering and then feedforward ANN for selecting the best solution for each cluster. In their procedures (77; 78), the decision-maker is actively involved by comparing and contrasting solutions. (24) extended the clustering approaches formalizing the concept of k representative points of the Pareto front, where Pareto optimal solutions are clustered, and then the Pareto frontier is divided into k

clusters. The k-means algorithms are used in (120; 24) for clustering. Recently, a graph-theoretical clustering approach was proposed for finding a reduced set of Pareto optimal solutions (63), where they construct a contact network by mapping each point in the objective space to a node, and connecting nodes that are within a certain distance of each other. One way to use the idea of clustering is to use an off-the-shelf clustering algorithm to cluster PF. Within a cluster $CL_h \subseteq PF$, we choose the status set $SS[h] \in CL_h$ that minimizes the distance to the other members of the cluster (according to a selected distance metric). Such a status set would be like a pseudo-centroid for that cluster. We can then create a graph whose nodes are these pseudo-centroids and whose edges are labeled according to the distance metric and choose the pseudo-centroid with the highest centrality (e.g., betweenness centrality (19) or Pagerank (20)).

4.9. Limitations and Future Work

We now describe a few limitations of our work that can also lead to potential future work.

Scalability. In the real-world, there can be thousands of agents (e.g., thousands of cars on a single highway or road at a given point in time) and the responses to the actions of other cars has to be done at lightning speed. While the experiments show that POSS can be solved in 200 milliseconds to 1 second in several cases, there are some important cases where the computation time can be a few seconds. Fast approximation algorithms that provide solutions within milliseconds, yet are guaranteed to be within some approximation error bound of the optimal solution, need to be developed.

Scalable Choice of a Pareto-Optimal Status Set. One of the strengths of this chapter is that we can find a Pareto-Optimal Status Set without computing the entire Pareto frontier. Though we have outlined some methods to choose from a Pareto frontier in Section 4.8, it is important to adapt our proposed algorithms to find such Pareto-Optimal Status Sets without computing the entire Pareto frontier as that could compromise scalability. We also need to look at methods to extend the approximation algorithms mentioned above to this case.

Error-Tolerance. When multiple agents are operating in the real world, there will be noise and errors, e.g., errors due to sensor malfunction and/or due to communication latency or dropped packets between agents. What does it mean for a Pareto-Optimal Feasible Status Set to be robust to some kind of noise or error? How should our algorithms be changed in order to achieve such robustness without compromising scalability? It is critical to investigate this question further.

Long-Horizon Decision Making. Our proposed approach can effectively find a POSS for the next time step. It is worth investigating how to extend our approach for long-horizon decision making to improve the overall quality of the solution, considering potential actions of other agents, but without compromising scalability.

4.10. Conclusions

In this chapter, we have developed the concept of a multi-agent system in which multiple agents each try to optimize multiple objectives in accordance with an input set of behavioral models and objectives. We specified the behavioral constraints in a high-level deontic logic, so that users and application developers can express their desired

logical constraints easily in symbolic form, while simultaneously expressing their objective functions numerically. Our agents can work with *any* behavior model expressed in the deontic logic used here and any set of objective functions.

This chapter makes several novel contributions. To the best of our knowledge, this is the first work to consider multiple objective functions when deciding what actions a deontic logic agent should take. Second, we are the first to show co-NP-hardness of deciding whether a given status set is a POSS. Third, we are the first to develop (multiple) algorithms for solving the POSS problem both exactly as well as approximately under varying assumptions on the form of the objective functions and conducted an extensive set of experiments.

While there are opportunities for future research as discussed in Section 4.9, our work represents a first contribution to the science that integrates deontic logic for high level reasoning and Pareto optimization methods for lower-level reasoning, which can be applied in several real-world applications involving multiple agents.

Algorithm 5 POSS weakly-monotonic-approximate

Input: A state S_t, an agent program P,
a set IC of integrity constraints,
a set AC of action constraints, a conc function,
a set OF of weakly monotonic objective functions,
a set A of ground actions, and
an integer τ (number of samples for randomly picking).
Output: A feasible status set or $\bot$.
Let DC be the set of denial constraints in AC.
$LSS = \mathsf{Closure}(\emptyset, S_t, P, DC)$.
if $LSS = \bot$ **then**
  return $\bot$.
end if
$\overline{A} := \{\alpha \mid \alpha \in A$ and $(Pre(\alpha)$ *is false in* S_t or $\mathbf{F}\alpha \in LSS)\}$.
$ToInspect := \{LSS \cup \{\mathbf{F}\alpha \mid \alpha \in \overline{A}\}\}$.
for each $\alpha \in (A \setminus \overline{A})$ **do**
  $Tmp := \emptyset$.
  for each $SS \in ToInspect$ **do**
    if $\mathbf{P}\alpha \notin SS$ **then**
      Add $SS \cup \{\mathbf{F}\alpha\}$ to Tmp.
    end if
    if $\mathbf{F}\alpha \notin SS$ **then**
      Add $SS \cup \{\mathbf{O}\alpha, \mathbf{Do}\alpha, \mathbf{P}\alpha\}$ to Tmp.
    end if
  end for
  $ToInspect := Tmp$.
end for
$Done := \emptyset$.
while $ToInspect \neq \emptyset$ **do**
  $Candidates := random(\tau, ToInspect)$.
  $ToInspect := ToInspect \setminus Candidates$.
  if *Candidates* has a feasible status set **then**
    return a Pareto-optimal (w.r.t. OF) feasible status set of *Candidates*.
  else
    for each *Cand* in *Candidates* **do**
      for each $Op\,\alpha \in Cand$ **do**
        if $Op\,\alpha \notin LSS$ and $(Cand \setminus \{Op\,\alpha\}) \notin ToInspect$ and $(Cand \setminus \{Op\,\alpha\}) \notin Done$ **then**
          if $Op = \mathbf{O}$ or $Op = \mathbf{F}$ **then**
            Add $Cand \setminus \{Op\,\alpha\}$ to $ToInspect$.
          end if
          if $Op = \mathbf{Do}$ and $\mathbf{O}\alpha \notin Cand$ **then**
            Add $Cand \setminus \{Op\,\alpha\}$ to $ToInspect$.
          end if
          if $Op = \mathbf{P}$ and $\mathbf{O}\alpha \notin Cand$ and $\mathbf{Do}\alpha \notin Cand$ **then**
            Add $Cand \setminus \{Op\,\alpha\}$ to $ToInspect$.
          end if
        end if
      end for
    end for
    Add *Candidates* to *Done*.
  end if
end while
return $\bot$.

Algorithm 6 POSS strongly-monotonic-approximate

Input: A state S_t, an agent program P,
a set IC of integrity constraints,
a set AC of action constraints, a conc function,
a set OF of strongly monotonic objective functions,
a set A of ground actions, and
an integer τ (number of samples for randomly picking).
Output: A feasible status set or $\bot$.
Let DC be the set of denial constraints in AC.
$LSS = \mathsf{Closure}(\emptyset, S_t, P, DC)$.
if $LSS = \bot$ **then**
  return $\bot$.
end if
$\overline{A} := \{\alpha \mid \alpha \in A$ and ($Pre(\alpha)$ *is false in* S_t or $\mathbf{F}\alpha \in LSS$)$\}$.
$A' := A \setminus \overline{A}$.
$SA\text{-}DPO := \bigcup_{\alpha \in A'} \{\mathbf{Do}\alpha, \mathbf{P}\alpha, \mathbf{O}\alpha\}$.
$SA\text{-}F := \{\mathbf{F}\alpha \mid \alpha \in \overline{A}\}$.
$ToInspect := \{LSS \cup SA\text{-}DPO \cup X \mid X \subseteq SA\text{-}F\}$.
$Done := \emptyset$.
while $ToInspect \neq \emptyset$ **do**
  $Candidates := random(\tau, ToInspect)$.
  $ToInspect := ToInspect \setminus Candidates$.
  if *Candidates* has a feasible status set **then**
    return a Pareto-optimal (w.r.t. OF) feasible status set of *Candidates*.
  else
    for each *Cand* in *Candidates* **do**
      for each $\mathbf{Do}\alpha \in Cand$ **do**
        if $\mathbf{Do}\alpha \notin LSS$ **then**
          if $(Cand \setminus \{\mathbf{Do}\alpha, \mathbf{P}\alpha, \mathbf{O}\alpha\}) \notin ToInspect$ and $(Cand \setminus \{\mathbf{Do}\alpha, \mathbf{P}\alpha, \mathbf{O}\alpha\}) \notin Done$ **then**
            Add $Cand \setminus \{\mathbf{Do}\alpha, \mathbf{P}\alpha, \mathbf{O}\alpha\}$ to *ToInspect*.
          end if
          if $(Cand \setminus \{\mathbf{Do}\alpha, \mathbf{O}\alpha\}) \notin ToInspect$ and $(Cand \setminus \{\mathbf{Do}\alpha, \mathbf{O}\alpha\}) \notin Done$ **then**
            Add $Cand \setminus \{\mathbf{Do}\alpha, \mathbf{O}\alpha\}$ to *ToInspect*.
          end if
          if $((Cand \cup \{\mathbf{F}\alpha\}) \setminus \{\mathbf{Do}\alpha, \mathbf{P}\alpha, \mathbf{O}\alpha\}) \notin ToInspect$ and $((Cand \cup \{\mathbf{F}\alpha\}) \setminus \{\mathbf{Do}\alpha, \mathbf{P}\alpha, \mathbf{O}\alpha\}) \notin Done$ **then**
            Add $(Cand \cup \{\mathbf{F}\alpha\}) \setminus \{\mathbf{Do}\alpha, \mathbf{P}\alpha, \mathbf{O}\alpha\}$ to *ToInspect*.
          end if
        end if
      end for
    end for
    Add *Candidates* to *Done*.
  end if
end while
return $\bot$. =0

CHAPTER 5

GUARDIAN: Governance-Unified Aerial Reinforcement-Defense In Accordance with Norms

Previous chapters established capabilities for threat prediction (Chapter 2), data generation (Chapter 3), and legally compliant decision-making (Chapter 4). However, urban drone defense requires not only identifying optimal responses to known threats but also adaptive learning of effective strategies as adversarial tactics evolve. Consider a drone swarm attack by an adversary (RED) who targets a city C defended by a player (BLUE), where BLUE must comply with various legal and ethical requirements that RED ignores. Will such compliance requirements negatively impact BLUE's defense of C? We propose GUARDIAN (Governance-Unified Aerial Reinforcement-Defense In Accordance with Norms), a novel decision-making framework that integrates reinforcement learning (RL) with deontic logic, which enables BLUE to determine how a set of legal/ethical norms influences the defense of a target city attacked by a set of drones. We extensively evaluate GUARDIAN by varying city size, the number of drones, the ratio of BLUE vs. RED drones, and other parameters. Surprisingly, our experiments show that BLUE's defense of C may not always be compromised by the requirement for compliance, even if RED attacks in violation of the norms. As current conflicts increasingly involve drones, GUARDIAN addresses the challenge of ethical compliance in RL-based decision making

in defense scenarios and of building trust in autonomous systems, towards ethical and secure deployment of autonomous defense systems.

5.1. Introduction

Drones have become a major part of warfare and terrorism. Terror groups such as Hezbollah (57) have operated drones since 2004 (115). ISIS, the Badr Brigade and armed groups in Libya (106) have also invested in drone warfare. Drones were used in a terror attack on the Jammu Air Force Station in India in June 2021.[1] (5) describes drone use by several groups, including ISIS, Hayat Tahrir al Sham in Syria, the Houthis, the PKK, Boko Haram, Marawi militants in the Philippines, and many others. The May 2025 India-Pakistan conflict saw the use of drones.[2] The ongoing conflict in Ukraine has seen numerous drone attacks on cities.[3]

There has also been considerable discussion on how drone warfare might violate the Laws of War (126) or International Humanitarian Law (42). Hence, it is critical to answer the following question: When one side (BLUE) is defending a city C (or region) using a drone swarm that complies with a set of norms, and another side (RED) is attacking C without complying with those norms, how disadvantaged is BLUE?

Figure 5.1 illustrates this challenge. A defending drone (B1) encounters an attacking drone (R1) that has infiltrated a high-value civilian area containing critical infrastructure. The defender faces a decision shaped by two competing imperatives: neutralizing the threat and avoiding civilian harm.

[1] https://www.msn.com/en-in/news/newsindia/use-of-drones-by-terror-groups-an-add-on-to-worries-india-at-unga/ar-AALzkhl?ocid=uxbndlbing
[2] https://www.bbc.com/news/articles/cwy6w6507wqo
[3] https://www.understandingwar.org/backgrounder/russian-drone-innovations-are-likely-achieving-effects-battlefield-air-interdiction

To answer this question, we developed GUARDIAN, a testbed for AI-enabled drone swarms. This chapter makes the following contributions:

- GUARDIAN assumes that a BLUE (resp., RED) headquarters (HQ) operates a set of BLUE (resp., RED) drones.
- Deontic logic (50) was invented ins (79) to express conditions under which an action is permitted, forbidden, or obligatory. Since then, it has been extensively used to express laws and regulations (53; 81). GUARDIAN's BLUE drones combine deontic logic with Reinforcement Learning (RL) to combat RED. This enables military planners to use GUARDIAN to plan operations that comply with relevant laws/ethics. To our knowledge, this is one of the first combinations of drone swarms, deontic logic, ethics, and RL in security settings. RED has no compliance requirements, but also uses RL.
- We extensively evaluate GUARDIAN by varying city size, the number of drones, and the ratio of BLUE vs. RED drones, as well as other parameters. Our experiments show that the requirement to comply with norms is not necessarily a disadvantage for BLUE. While it is not surprising that when RED drones are the majority, BLUE generally does not win, when the ratio BLUE/RED drones is higher, the city protection achieved is almost as effective as when there are no norms to be satisfied.

5.2. Related Work

Recent research (4; 133; 7) has explored methods to ensure that RL agents adhere to safety constraints while optimizing rewards, which involve synthesizing a reactive "shield"

to enforce properties specified in temporal or probabilistic logic. This shield monitors the actions of the RL agent and intervenes only when a chosen action would violate the specified constraints, thereby ensuring compliance with safety requirements during both learning and execution phases. (87) devised defeasible deontic logics to specify and enforce normative behavior in RL agents. (88) also introduced a normative supervisor module that embeds a theorem prover for defeasible deontic logic within the RL control loop. The supervisor acts both as an event recorder and as a real-time compliance checker to ensure alignment with ethical prescriptions. (110) proposed a probabilistic deontic logic to specify the obligations of stochastic systems, including ethical responsibilities. Their logic tries to align with the semantics of MDPs. (111) further proposed Expected Act Utilitarian deontic logic which facilitates reasoning about an agent's ethical and social obligations, permissions, and prohibitions at design time. It enables designers to identify conflicts between an agent's actions and its normative constraints. This approach manages trade-offs between mission objectives and ethical considerations, operating at the logical level rather than solely at the reward level.

Instead of only considering the behavior of a single agent, GUARDIAN integrates RL with deontic logic within multiagent environments. It appears to be the first system to do so in a military drone swarm cooperation-competition environment.

5.3. The GUARDIAN Framework

GUARDIAN involves two teams (BLUE and RED) with drones flying over a city C which is represented as an $(M \times N)$ grid. Each cell (i, j) (for $i \in [1, M], j \in [1, N]$) has a

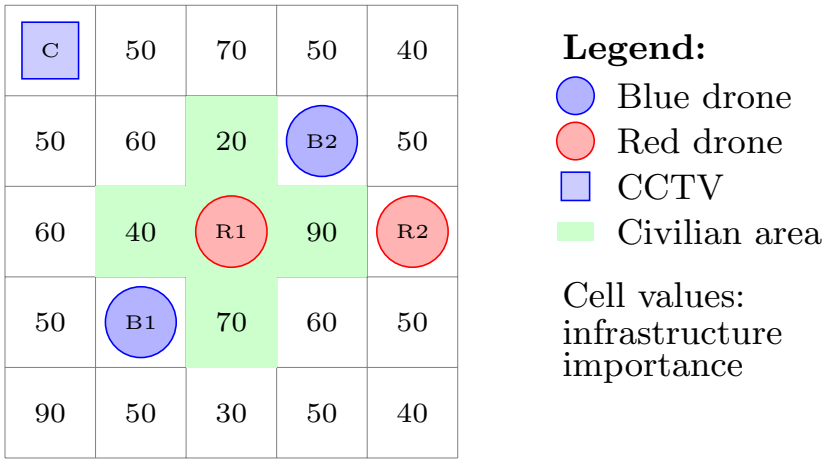


Figure 5.1. Example 1: A 5×5 urban defense scenario. Green cells represent civilian areas with restrictions on engagement. Cell values indicate infrastructure importance. At this time step, RED drone R1 has infiltrated the highest-value civilian area while BLUE drones B1 and B2 must decide their response under ethical constraints.

value $v_{i,j}(t) \in \mathbb{R}_{\geq 0}$ at any time t. Cell (i, j) is *alive* at time t if $v_{i,j}(t) > 0$ — otherwise it is *dead*. The value of the city at time t is $V(t) = \Sigma_{i\in[1,M],j\in[1,N]} v_{i,j}(t)$.

BLUE defends city C using a set of BLUE drones. All drones, BLUE and RED, can autonomously fly, capture/send imagery to HQ, sense other drones, and fire at them. BLUE also has CCTV cameras for surveillance. BLUE HQ uses captured data from the drones and CCTV cameras for decision making. *BLUE must abide by a given set of ethical norms. RED* operates drones and an HQ with the same actions as BLUE with the goal of destroying city C. In addition, RED drones can fire at cell (i, j). RED HQ uses data from RED drones to compute optimal strategies to maximize damage to BLUE (city and drones). *But RED does not follow any ethical norms.*[4] Formal definitions of these actions are reported in Appendix C.1.4

[4]This represents reality as terrorists do not follow ethical/legal norms.

5.3.1. Motivating Example

We use a simplified urban defense scenario throughout this chapter to illustrate key concepts. Figure 5.1 shows a 5×5 grid (though experiments use up to 128×128). Green cells indicate civilian areas, values represent infrastructure importance. Two defending BLUE drones must protect against Two attacking RED drones. While RED drones operate without ethical constraints, BLUE drones must comply with rules of engagement.

At time step t, BLUE drone B1 observes RED drone R1 occupying the central high-value civilian cell. This scenario illustrates the core challenge: how can B1 effectively defend critical infrastructure while respecting ethical constraints? We formalize ethical constraints using two deontic norms:

- Example Norm A: Forbid firing in civilian areas unless facing immediate threat
- Example Norm B: Obligate engagement when threats endanger high-value neighbors
- Example Norm C: Forbid firing at cells (only RED can destroy city infrastructure)

These norms generate multiple feasible actions for B1, creating a branching decision space that balances tactical effectiveness with ethical compliance. Hence, even in this small example, there are numerous feasible actions. The deontic logic layer (discussed in the next subsection) ensures only ethically and/or legally compliant actions remain available to drones.

Notably, both B1 and B2 could potentially engage R1 since it occupies the highest-value cell. However, the BLUE headquarters, which aggregates observations from both drones and the CCTV camera, recognizes that optimal defense requires coordinated action. The HQ computes joint strategies and suggests B1 engage R1 while recommending

B2 move toward R2 at position (1,5). This coordination prevents redundant targeting and ensures comprehensive threat coverage. Each drone validates the HQ's suggestion against its plan before execution, maintaining ethical compliance while achieving tactical coordination.

5.3.2. Deontic Ethical/Legal Norms

Our deontic logic has two kinds of logical atoms: *state atoms* and *action atoms*. State atoms describe the current state of the environment and drones, while action atoms describe actions that drones can perform. Actions have preconditions and effects detailed in Appendix C.1.5 and C.1.6.

State Predicates for Drones.

- $blue(d)$ / $red(d)$: Drone d is on the BLUE (defending) or RED (attacking) Team.
- $position(d, i, j, t)$: Drone d is located in cell (i, j) at time t. Formally, this predicate holds iff $\mathbf{x}_d(t) = (i, j)$, where $\mathbf{x}_d(t)$ is the position vector defined in Appendix C.1.
- $InFireRange(d, d')$: Drone d' is within drone d's firing range, i.e., $\|\mathbf{x}_d(t) - \mathbf{x}_{d'}(t)\| \leq f_d$.
- $HasPayload(d)$: Drone d has nonzero ammunition remaining, i.e., $p_d(t) > 0$.
- $ImmediateThreat(d')$: Drone d' is deemed an immediate threat. This holds when the observed public state of d' satisfies $B_{d'} > 0$ (operational) and $p_{d'} > 0$ (armed).
- $SameTeam(d, d')$: Drones d and d' belong to the same team.

State Predicates for Environment.

- $CivilianArea(i, j)$: Cell (i, j) is designated as a protected civilian region.

- *Adjacent*(i, j, p, q): Cell (p, q) is orthogonally adjacent to (i, j), i.e., $|i - p| + |j - q| = 1$ and both cells are within grid bounds.

State Predicates for HQ Communication.

- *CommConsistent*(d, t): Communication between drone d and its HQ is functioning at time t (modeled as Bernoulli with probability p_{comm}).
- *AssignedAction*(d, a, t): The HQ has assigned action a to drone d at time t.

Derived Utility Predicates. Certain deontic norms compare the utility of neighboring cells to determine engagement obligations. We introduce two threshold parameters λ and λ' where $1 < \lambda < \lambda'$, chosen by domain experts (e.g., $\lambda = 1.1$, $\lambda' = 2.0$).

- *HasLowerUtilityNeighbor*(i, j, t, λ): Cell (i, j) has at least one neighbor with strictly lower utility, and no neighbor exceeds $\lambda \cdot v_{i,j}(t)$:

$$\exists (p, q) : \mathit{Adjacent}(i, j, p, q) \wedge v_{p,q}(t) < v_{i,j}(t),$$

$$\forall (p', q') : \mathit{Adjacent}(i, j, p', q') \Rightarrow v_{p',q'}(t) < \lambda\, v_{i,j}(t).$$

  Intuition: If a RED drone is not immediately threatening and might move to a less valuable cell, BLUE should wait rather than engage.

- *AllNeighborsAbove*(i, j, t, λ): Every neighbor of (i, j) has utility $\geq \lambda \cdot v_{i,j}(t)$:

$$\forall (p, q) : \mathit{Adjacent}(i, j, p, q) \Rightarrow v_{p,q}(t) \geq \lambda\, v_{i,j}(t).$$

  Intuition: If all surrounding cells are more valuable, allowing the RED drone to move risks greater damage.

- *HighValueNeighbor*(i, j, t, λ'): At least one neighbor of (i, j) has utility $\geq \lambda' \cdot v_{i,j}(t)$:

$$\exists (p, q) : Adjacent(i, j, p, q) \wedge v_{p,q}(t) \geq \lambda' \, v_{i,j}(t).$$

  Intuition: Even if most neighbors are not critical, one extremely high-value neighbor demands immediate engagement.

Action Predicates.

- *FireAtDrone*$_d(d')$: Drone d fires at drone d'.
- *FireAtCell*$_d(i, j)$: Drone d fires at cell (i, j) to destroy infrastructure.
- *MoveTo*$_d(i, j)$: Drone d moves to cell (i, j).
- *ExecuteAssignedAction*$_d(a)$: Drone d executes HQ-assigned action a.

We do not claim new contributions to deontic logic in this chapter. We build on (44; 45). Our contribution is in the ***combination*** *of deontic logic and Reinforcement Learning and their application to protecting cities from drone swarm attacks.*

A *status atom* in GUARDIAN is an expression of the form $\mathbf{P}a$, $\mathbf{O}a$, $\mathbf{F}a$, or $\mathbf{Do}\, a$, indicating that action atom a is permitted, obligatory, forbidden, or to be done, respectively. A *status set* is a set of status atoms. A *Deontic Rule* (DR for short) has the form

$$SA \leftarrow \chi \;\&\; SA_1 \;\&\; \ldots \;\&\; SA_n$$

where $SA, SA_1, \ldots, SA_n$ are status atoms and χ is a conjunction (logical AND) of state predicates. BLUE drones have an associated set $\mathcal{N}_d$ of DRs.

GUARDIAN uses eight deontic norms (expressed as Deontic Rules) obtained in consultation with security experts:

DR 1: Never firing at cell. A BLUE drone is always forbidden from deliberately firing on a cell (i, j). Only RED drones may destroy city infrastructure.

DR 2: Prohibition of firing at civilian areas. The BLUE drone must refrain from firing at a RED drone in a cell (i, j) designated as a civilian area, unless the RED drone is an immediate threat. For example, if both drones are co-located in a hospital cell and the RED drone is not an immediate threat, the BLUE drone is forbidden from engaging.

DR 3: Obligation to follow HQ orders (if communication is consistent). The BLUE drone must comply with HQ instructions when the communication channel is reliable, ensuring centralized coordination.

DR 4: Prohibition of friendly fire. A BLUE drone must never fire at another BLUE drone.

DR 5: Permission to engage a RED drone in a civilian area (under threat). If a RED drone is an *immediate threat* inside a civilian area, the BLUE drone *may* fire to prevent severe harm. This overrides DR 2 when threat conditions are met.

DR 6: Forbid firing if RED drone is not an immediate threat and a lower-utility neighbor exists. If the RED drone is not evidently threatening ($\neg ImmediateThreat(d')$) and cell (i, j) has a lower-utility neighbor, the BLUE drone should wait, hoping the RED drone relocates to the less valuable cell.

DR 7: Obligated to engage a threat if all neighbors are higher utility. If all neighboring cells are more valuable than (i, j) and the RED drone is an immediate threat, the BLUE drone *must* fire immediately to prevent the threat from moving to more critical cells.

DR 8: Obligated to engage a threat if any neighbor is extremely high-value. If any neighboring cell is extremely critical (utility $\geq \lambda' \cdot v_{i,j}(t)$) and the RED drone is an immediate threat, the BLUE drone must fire to prevent catastrophic damage.

The formal specification using deontic operators is:

Formal Specification of Norms $\mathcal{N}_d$ as Deontic Rules

Norm 1: Never firing at cell: $\mathbf{F}\mathit{FireAtCell}_d(i,j) \leftarrow \mathit{blue}(d)$

Norm 2: Prohibition of firing at civilian areas: $\mathbf{F}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{CivilianArea}(i,j) \wedge \mathit{InFireRange}(d,d') \wedge \neg \mathit{ImmediateThreat}(d')$

Norm 3: Obligation to follow HQ orders: $\mathbf{O}\mathit{ExecuteAssignedAction}_d(a) \leftarrow \mathit{blue}(d) \wedge \mathit{CommConsistent}(d,t) \wedge \mathit{AssignedAction}(d,a,t)$

Norm 4: Prohibition of friendly fire: $\mathbf{F}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{blue}(d') \wedge \mathit{SameTeam}(d,d')$

Norm 5: Permission to engage under threat: $\mathbf{P}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{InFireRange}(d,d') \wedge \mathit{CivilianArea}(i,j) \wedge \mathit{ImmediateThreat}(d')$

Norm 6: Forbid firing if lower-utility escape exists: $\mathbf{F}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{InFireRange}(d,d') \wedge \neg \mathit{ImmediateThreat}(d') \wedge \mathit{HasLowerUtilityNeighbor}(i,j,t,\lambda)$

Norm 7: Obligated engagement (all neighbors higher): $\mathbf{O}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{InFireRange}(d,d') \wedge \mathit{ImmediateThreat}(d') \wedge \mathit{AllNeighborsAbove}(i,j,t,\lambda)$

Norm 8: Obligated engagement (critical neighbor): $\mathbf{O}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{InFireRange}(d,d') \wedge \mathit{ImmediateThreat}(d') \wedge \mathit{HighValueNeighbor}(i,j,t,\lambda')$

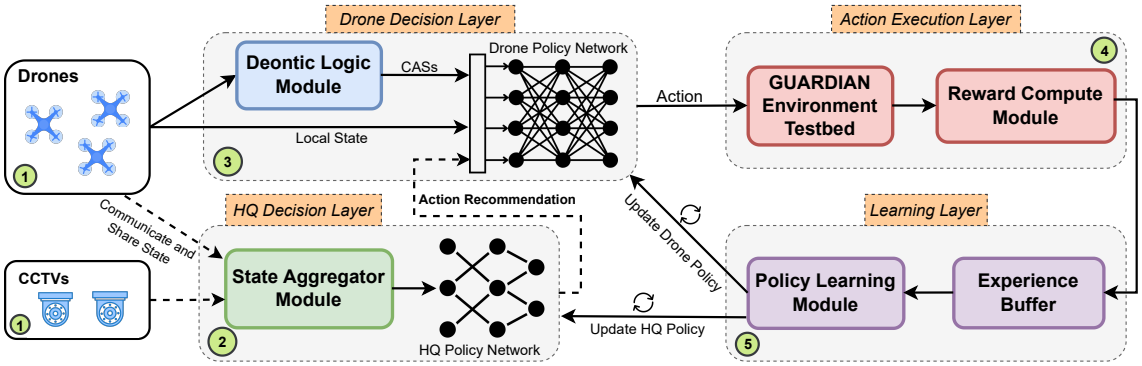


Figure 5.2. Overview of the GUARDIAN architecture. Drones and CCTVs (1) share state information with both individual decision modules (2) and headquarters (3). Each drone's deontic logic module computes feasible actions (CASs) to ensure ethical compliance before the policy network makes decisions. The HQ aggregates global state and provides coordination recommendations, but drones retain autonomy to validate these against feasible status sets. Actions execute in the environment (4), generating rewards that feed into the learning layer (5) to update both individual drone and HQ policy networks. This design enables centralized training with decentralized, ethically-constrained execution.

In a given state s at time t, drone d computes a set $\mathcal{F}_d(s)$ of *feasible status sets* which are guaranteed to satisfy all norms (44; 45):

$$\mathcal{F}_d(s(t)) = \{SS_d \mid SS_d \text{ is a status set feasible in state } s(t)\}. \tag{5.1}$$

We write $\mathcal{F}_d(s)$ when the time index is clear from context.

Each $SS_d \in \mathcal{F}_d(s)$ contains some subset of $\mathbf{Do}\alpha_d$ status atoms, where α_d is a ground action for drone d. Hence, each *feasible status set* SS_d implicitly represents a *concurrent action set (CAS)*

$$X_{SS_d} = \{\alpha_d \mid \mathbf{Do}\alpha_d \in SS_d\}. \tag{5.2}$$

Each X_{SS_d} satisfies all DRs by results in (44).

Returning to our motivating example (Figure 5.1), consider BLUE drone B1 and RED drone R1 in the same cell (3,3). Let us assume R1 has payload remaining (so it is an immediate threat). In this state, B1's deontic logic module computes the following feasible status sets (FSS):

- **FSS-1:** $\{\mathbf{P}(\mathit{FireAtDrone}_{B1}(R1)), \mathbf{Do}(\mathit{FireAtDrone}_{B1}(R1))\}$

  Rationale: Since R1 is an immediate threat in a civilian area, Example Norm A permits engagement. This status set chooses to execute the fire action.
- **FSS-2:** $\{\mathbf{P}(\mathit{MoveTo}_{B1}(\text{up})), \mathbf{Do}(\mathit{MoveTo}_{B1}(\text{up}))\}$

  Rationale: Moving to cell $(2, 3)$ is always permitted. This status set chooses to reposition rather than engage.
- **FSS-3:** $\{\mathbf{P}(\mathit{MoveTo}_{B1}(\text{down})), \mathbf{Do}(\mathit{MoveTo}_{B1}(\text{down}))\}$

  Rationale: Moving to cell $(4, 3)$ is permitted. This represents another repositioning option.
- **FSS-4:** $\{\mathbf{P}(\mathit{FireAtDrone}_{B1}(R1)), \mathbf{P}(\mathit{MoveTo}_{B1}(\text{left})), \mathbf{Do}(\mathit{MoveTo}_{B1}(\text{left}))\}$

  Rationale: Although firing is permitted, this status set chooses to move left to cell $(3, 2)$ instead.

In contrast, consider an **infeasible status set**: $\{\mathbf{Do}(\mathit{FireAtCell}_{B1}(3, 4))\}$ is infeasible because it violates Example Norm C, which forbids BLUE drones from firing at cells. No feasible status set can include this action.

Now suppose R1 had *no* payload remaining (i.e., not an immediate threat). In this case: $\{\mathbf{Do}(\mathit{FireAtDrone}_{B1}(R1))\}$ would be **infeasible** because Example Norm A forbids firing in the civilian area. The status set would contain $\mathbf{F}(\mathit{FireAtDrone}_{B1}(R1))$, making any set with $\mathbf{Do}(\mathit{FireAtDrone}_{B1}(R1))$ infeasible.

This example illustrates how the same physical state can yield different feasible status sets depending on threat assessment, and how ethical norms constrain the action space available to BLUE drones.

5.3.3. Feasible Status Set Computation

We present the procedure for computing drone d's feasible status sets. Our approach relies on two algorithms from (36): (1) Least Status Set (LSS) generation, and (2) Ethical Status Set enumeration. Throughout these algorithms, we write $Pre(\alpha_d)$ to denote the preconditions of action α_d, as formally defined in Appendix C.1.5.

The computation also enforces two types of constraints:

- **Integrity Constraints (IC)**: Conditions ensuring system consistency, such as requiring a target to be within firing range before engagement.
- **Action Constraints (AC)**: Rules governing permissible action combinations, such as preventing a drone from engaging multiple targets simultaneously.

Formal definitions of these constraints are provided in Appendix C.2.

If LSS returns $\perp$, no ethically compliant action exists; the drone executes no-op. Otherwise, each feasible status set SS_d yields a concurrent action set $X_{SS_d} = \{\alpha_d \mid \mathbf{Do}\alpha_d \in SS_d\}$.

5.4. Combining Deontic Logic with RL

BLUE's actions must comply with its ethical norms. In standard RL formulations, a drone's *action space* $\mathcal{A}_d$ has *single* actions. However, in GUARDIAN, *each* CAS X_{SS_d} (containing multiple actions) is a *single action* from the RL perspective that drone d

Algorithm 7 *LSS*: Least Status Set Algorithm for Drone d

Input: Status set SS, drone state $S_d(t)$, norms $\mathcal{N}_d$, denial constraints DC
Output: A status set SS_d or $\perp$

for each $\mathbf{O}\alpha_d \in SS$ s.t. $\mathbf{P}\alpha_d \notin SS$ **do**
    Add $\mathbf{P}\alpha_d$ to SS
end for
for each $\mathbf{O}\alpha_d \in SS$ s.t. $\mathbf{Do}\alpha_d \notin SS$ **do**
    Add $\mathbf{Do}\alpha_d$ to SS
end for
for each $\mathbf{Do}\alpha_d \in SS$ s.t. $\mathbf{P}\alpha_d \notin SS$ **do**
    Add $\mathbf{P}\alpha_d$ to SS
end for
if $\exists \alpha_d$: $\{\mathbf{P}\alpha_d, \mathbf{F}\alpha_d\} \subseteq SS$ **or** $\mathbf{P}\alpha_d \in SS$ with false preconditions **then return** $\perp$
end if
if $\{\alpha_d \mid \mathbf{Do}\alpha_d \in SS\}$ violates DC **then return** $\perp$
end if
$SS'_d := SS$
repeat
    $SS''_d := SS'_d$
    for each rule $r : SA_d \leftarrow \chi \,\&\, SA_{d,1} \,\&\, \ldots$ in $\mathcal{N}_d$ **do**
        if χ true in $S_d(t)$ **and** $\{SA_{d,1}, \ldots\} \subseteq SS'_d$ **then**
            Add SA_d to SS'_d; propagate $\mathbf{O} \Rightarrow \mathbf{P}, \mathbf{Do}$
            if contradiction or constraint violation detected **then return** $\perp$
            end if
        end if
    end for
until $SS'_d = SS''_d$ **return** SS'_d

Algorithm 8 Ethical Status Set Computation for Drone d

Input: HQ suggestions SS_{HQ}, state $S_d(t)$, norms $\mathcal{N}_d$, constraints IC, AC, threshold τ
Output: Set of feasible status sets $\{SS_d\}$ or $\perp$

$DC \leftarrow$ denial constraints from AC
$LSS_d \leftarrow \mathrm{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$
if $LSS_d = \perp$ **then**
    $LSS_d \leftarrow \mathrm{LSS}(\emptyset, S_d(t), \mathcal{N}_d, DC)$
    if $LSS_d = \perp$ **then return** $\perp$ // No compliant actions exist
    end if
end if
Identify forbidden/infeasible actions $\overline{A}_d$
Build candidate sets by extending LSS_d with permitted **Do** atoms
Enumerate up to τ feasible sets satisfying IC and AC **return** feasible status sets

chooses in the current state s. Formally,

$$\mathcal{A}_d(s) \;=\; \big\{X_{SS_d} \;\mid\; SS_d \in \mathcal{F}_d(s)\big\}. \tag{5.3}$$

All status sets outside $\mathcal{F}_d(s)$ either violate ethical norms or feasibility. This *masks out* all status sets that violate one or more DRs.

Pruning the Exponential Space. If there are $|\mathcal{A}_d|$ possible ground actions for drone d, then up to $2^{|\mathcal{A}_d|}$ CASs are theoretically possible. However, the definition of FSSs (Appendix C.2) excludes most subsets, drastically shrinking the action space. We leverage the feasible status set computation algorithm from our previous work (36) (detailed in Appendix C.2.6) to efficiently enumerate only ethically compliant action sets.. This ensures that drone d only explores ethically valid CASs in its learning process.

5.4.1. Ethically/Legally-Guided MDPs

We now explain how GUARDIAN integrates *feasible status sets* into an RL framework so that BLUE drones (and similarly, BLUE HQ) can learn *optimal* policies satisfying all DRs. For simplicity, we focus on a single BLUE drone d — an analogous approach applies to other BLUE drones. We also describe a hierarchical approach where the BLUE/RED HQ operates at a higher strategic level to coordinate multiple drones.

BLUE drone d's Decision-making Problem can be framed within an MDP framework as follows. We treat d as an RL agent with an MDP $\mathcal{M}_d$, subject to an *action masking* mechanism that eliminates non-CASs.

5.4.1.1. MDP for Autonomous Drones. Each drone d uses an MDP:
$\mathcal{M}_d = \big(\mathcal{S}_d,\ \widehat{\mathcal{A}}_d,\ P_d,\ R_d,\ \gamma\big)$, where:

- **States.** $\mathcal{S}_d$ is the state space capturing drone d's observations. At time t, the drone's internal state is $s_d(t)$, which may include its position, health, resource levels, partial observations of the environment, etc.
- **Action Space.** $\widehat{\mathcal{A}}_d$ is the *masked* action space given by Eq. (5.3). In each state s, the *available* actions are exactly those CASs X_{SS_d} that satisfy all DRs for d at s.
- **Transition Function.** $P_d\big(s, X_{SS_d}, s'\big)$ is the probability of transitioning to state s' from state s when drone d executes all actions in X_{SS_d}. This accounts for:
  - Movement outcomes (e.g., boundary checks, success/failure probabilities).
  - Firing success or engagement outcomes, e.g., being able to attack and eliminate a RED drone successfully.
  - Concurrent effects of the environment (e.g., being attacked by an enemy drone) as per (44).
- **Reward Function.** $R_d(s, X_{SS_d}) \in \mathbb{R}$ is the immediate reward after executing the CAS X_{SS_d}. Unlike traditional single-action MDPs, the drone's reward depends on the *combined* effect of all $\alpha_d \in X_{SS_d}$.
- **Discount Factor.** $\gamma \in [0, 1]$ balances immediate and future rewards.

Concurrent-Action Rewards. We emphasize that $R_d(s, X_{SS_d})$ reflects the outcome *after* all actions in X_{SS_d} have been applied concurrently. For instance, if d chooses to $MoveTo_d$ to a new location *and* $FireAtDrone_d(d')$ simultaneously (assuming this satisfies the DRs), the reward for that step will incorporate the success/failure of both moving and firing. The reward formulation is discussed at Section 5.4.3 and Appendix C.2.7.

From our motivating example in Figure 5.1, consider drone B1 choosing between FSS-1 (fire at R1) and FSS-2 (move up). If B1 executes FSS-1 and eliminates R1: 1) B1 receives a kill bonus. 2) B1 incurs an ammunition cost. 3) The team receives an additional bonus for eliminating a threat.

If B1 instead executes FSS-2 (move up to cell $(2, 3)$): 1) B1 receives a survival bonus. 2) If this move brings B1 closer to the nearest enemy, B1 receives a movement incentive. 3) The team may incur future penalties if R1 subsequently destroys high-value cells.

The optimal policy balances immediate tactical gains against long-term strategic positioning, while ensuring all selected actions comply with ethical norms.

5.4.1.2. Policy Learning for Drones. A *policy* π_d for drone d is a mapping from states $s \in \mathcal{S}_d$ to probability distributions over CASs $X_{SS_d} \in \widehat{\mathcal{A}}_d(s)$. Formally:

$$\pi_d : \mathcal{S}_d \;\rightarrow\; \Delta\big(\widehat{\mathcal{A}}_d(s)\big),$$

where $\Delta(X)$ denotes the probability simplex over set X, i.e., the set of all probability distributions over X. The goal is to find an optimal policy

$$\pi_d^* \;=\; \arg\max_{\pi_d} \; \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \, R_d\big(s(t), X_{SS_d}(t)\big)\Big],$$

subject to the constraint that at each time t, the selected CAS $X_{SS_d}(t) \in \widehat{\mathcal{A}}_d(s(t))$ satisfies the deontic rules expressing ethical constraints. Here, $X_{SS_d}(t)$ denotes the CAS chosen at time t from the feasible status sets computed for state $s(t)$. In practice, one can use off-the-shelf RL algorithms (e.g., Q-learning, SARSA, policy gradient methods) (12) with an *action mask* that eliminates CASs not in $\widehat{\mathcal{A}}_d(s)$.

In the state shown in Figure 5.1, drone B1's policy π_{B1} assigns probabilities to each feasible status set (FSS-1 through FSS-4). After training, the policy learns to assign higher probability to FSS-1 (engage R1) when the expected reward from eliminating the threat outweighs the risk of leaving other areas undefended. The policy also considers coordination with other BLUE drones; if B2 is better positioned to engage R1, B1's policy may prefer FSS-2 (reposition) to cover another sector.

5.4.2. MDP for the Headquarters

The HQ operates at a higher level, coordinating multiple drones to achieve team-wide objectives. Let $\mathcal{D}_k$ be the set of drones under HQ k's command. The HQ has its own MDP:

$$\mathcal{M}_k^{\mathrm{HQ}} = \left(\mathcal{S}_k^{\mathrm{HQ}},\ \mathcal{A}_k^{\mathrm{HQ}},\ P_k^{\mathrm{HQ}},\ R_k^{\mathrm{HQ}},\ \gamma\right).$$

HQ State. The state $\mathcal{S}_k^{\mathrm{HQ}}(t)$ aggregates information from all drones under HQ k's control. Specifically, BLUE HQ's state includes:

- State of all BLUE drones.
- Observations from all CCTV cameras (positions and statuses of visible drones).
- Observed positions and public states of RED drones (within view range of any BLUE drone).
- Current status of the grid (which cells have been destroyed, remaining city value).

HQ Actions. The HQ's action space consists of *suggesting* actions to each drone:

$$\mathcal{A}_k^{\mathrm{HQ}}(t) = \prod_{d \in \mathcal{D}_k} \mathrm{SuggestedAction}_k(d),$$

where $\text{SuggestedAction}_k(d)$ is an action that the HQ *recommends* drone d execute. The actual execution depends on the drone's validation against its feasible status sets. $P_k^{\text{HQ}}(s, a, s')$ captures the outcome of finally executed actions by the drones and $R_k^{\text{HQ}}(s, a)$ reflects *team-level* outcome (e.g., total coverage of regions, elimination of enemy drones, discussed in Appendix C.2.7).

Hence, the HQ performs three key functions:

(1) **State Aggregation.** The HQ collects observations from all drones and CCTVs. Since individual drones have limited view ranges, the HQ provides a global picture by combining all local observations.

(2) **State Dissemination.** The HQ shares relevant global information with drones. When communication succeeds, drones receive updated global state. When communication fails, drones rely on cached information.

(3) **Action Coordination.** Based on the global state, the HQ computes suggested actions for each drone to maximize team-level objectives. For example in Figure 5.1, if both B1 and B2 can engage R1, the HQ might suggest B1 engage R1 while B2 repositions to cover R2, preventing redundant targeting.

Authority Overriding Drone Feasibility. HQs send *suggested* actions to drones; ethical norms are encoded at the drone level, not the HQ level. Hence, we do not include ethical norms for the HQs in this chapter. Each drone d must validate any suggestion from its HQ against its feasible status sets using Algorithms 7 and 8 before execution. If an HQ-suggested action conflicts with the BLUE drone's DRs (e.g., explicitly fire over a civilian area), then the drone will not take this action. Hence, even HQ-based policies cannot force a drone to violate its ethical constraints.

5.4.3. Reward Functions

We define reward functions that incentivize BLUE drones to protect the city while penalizing resource expenditure and rewarding threat elimination. RED drones receive symmetric but opposing rewards. Let $\mathcal{D}_{\text{BLUE}}(t)$ and $\mathcal{D}_{\text{RED}}(t)$ denote the sets of alive BLUE and RED drones at time t, respectively.

5.4.3.1. Notation.

- cost_d: Cost/value of drone d
- $B_d(t)$: Battery remaining for drone d at time t
- $p_d(t)$: Payload (ammunition) remaining for drone d at time t
- $v_c(t)$: Value of cell c at time t (equivalently $v_{i,j}(t)$ for cell (i, j))
- $\mathit{killed}(d, d', t)$: True if drone d eliminates drone d' at time t
- $\mathit{surv}(d, t)$: True if drone d survives time step t
- $\mathit{fired}(d, t)$: True if drone d fires at time t
- $\mathit{resp}(d, c, t)$: True if drone d is responsible for protecting cell c at time t
- $\mathit{alive}(c, t)$: True if cell c is alive (not destroyed) at time t

5.4.3.2. Reward Coefficients. The following coefficients balance different objectives:

- α: Weight for eliminating enemy drones (kill bonus)
- β: Weight for battery consumption (cost per time step)
- ζ: Weight for ammunition usage (cost per shot)
- δ: Weight for survival bonus
- ρ: Weight for protecting assigned cells
- ϕ: Weight for threat assessment (potential future damage)

5.4.3.3. Immediate Reward for BLUE Drone d.

$$r_t^d = \alpha \cdot \left(\sum_{\substack{d' \in \mathcal{D}_{\text{RED}}(t) \\ killed(d,d',t)}} \text{cost}_{d'} \right) - \beta \cdot \Delta B_d(t) - \zeta \cdot fired(d,t) \cdot \kappa_d$$
$$+ \delta \cdot surv(d,t) \cdot \sigma_d + \rho \cdot \left(\sum_{c:resp(d,c,t)} alive(c,t) \cdot v_c(t) \right)$$
$$- \phi \cdot \sum_{c' \in \mathcal{C}_{\text{danger}}(d',t)} v_{c'}(t) \cdot P_{\text{attack}}(d', c', t)$$

where:

- $\Delta B_d(t)$: Battery consumed by drone d during time step t
- κ_d: Ammunition cost coefficient for drone d
- σ_d: Survival bonus for drone d
- $\mathcal{C}_{\text{danger}}(d', t)$: Set of cells that RED drone d' can target given its current payload $p_{d'}(t)$ and battery $B_{d'}(t)$
- $P_{\text{attack}}(d', c', t)$: Estimated probability that d' attacks cell c'

5.4.3.4. Immediate Reward for RED Drone d'.

$$r_t^{d'} = \alpha \cdot \left(\sum_{\substack{d \in \mathcal{D}_{\text{BLUE}}(t) \\ killed(d',d,t)}} \text{cost}_d \right) - \beta \cdot \Delta B_{d'}(t) \tag{5.4}$$

$$- \zeta \cdot fired(d', t) \cdot \kappa_{d'} + \delta \cdot surv(d', t) \cdot \sigma_{d'}$$

$$+ \rho \cdot \left(\sum_{c: resp(d',c,t)} alive(c, t) \cdot v_c(t) \right)$$

$$+ \phi \cdot \sum_{c' \in \mathcal{C}_{\text{danger}}(d',t)} v_{c'}(t) \cdot P_{\text{attack}}(d', c', t)$$

Note the sign difference in the final term: BLUE is penalized for threat exposure while RED is rewarded.

5.4.3.5. Attack Probability Model. The probability that RED drone d' attacks cell c' within its remaining operational time is:

$$P_{\text{attack}}(d', c', t) = P_0(d', c', t) \cdot \left(1 - e^{-\mu \cdot B_{d'}(t)}\right)$$

where $\mu > 0$ is an attack urgency parameter and $P_0(d', c', t)$ is the base targeting probability:

$$P_0(d', c', t) = \frac{\left(\frac{v_{c'}(t)}{\|\mathbf{x}_{d'}(t) - \mathbf{x}_{c'}\| + \varepsilon}\right)^{\xi}}{\sum_{c'' \in \mathcal{C}_{\text{danger}}(d',t)} \left(\frac{v_{c''}(t)}{\|\mathbf{x}_{d'}(t) - \mathbf{x}_{c''}\| + \varepsilon}\right)^{\xi}}$$

This softmax-style distribution favors cells that are high-value and close to the RED drone. The parameter ξ (sharpness) controls selectivity, and ε prevents division by zero.

5.4.3.6. Team Reward. The team-level reward aggregates individual drone rewards:

$$R_t^{\text{BLUE}} = \sum_{d \in \mathcal{D}_{\text{BLUE}}(t)} r_t^d, \qquad R_t^{\text{RED}} = \sum_{d' \in \mathcal{D}_{\text{RED}}(t)} r_t^{d'}$$

5.4.4. Solving Ethically/Legally Guided MDPs

This section describes how computed FSSs (cf. Appendix C.2) are used in GUARDIAN to learn optimal or near-optimal policies for both *drones* and *Headquarters (HQs)*. By integrating the masking of violating CASs directly into the RL process (117; 6), each drone avoids violating ethical/legal norms. We adopt standard deep RL techniques for distributed agents (e.g., Deep Independent Q-Learning (121) for drones) and a centralized-coordination algorithm (QMIX (103)) for the Headquarters.

Independent Q-Learning for Drones. Each drone d applies Q-learning (or a deep variant) over its *masked* action space. Recall from Equation (5.3) that:

$$\widehat{\mathcal{A}}_d(s) \;=\; \big\{\, X_{SS_d} \;\mid\; SS_d \in \mathcal{F}_d(s) \big\}.$$

In the Q-learning context, this masking condition ensures that during both exploration (action sampling) and exploitation (greedy action selection), the Q-function only considers actions from $\widehat{\mathcal{A}}_d(s)$. Specifically, actions outside this set receive a mask value of $-\infty$ (or a large negative value), effectively excluding them from consideration.

Let $Q_d(s_d, X)$ denote drone d's Q-function, giving the expected discounted return for taking CAS X in local state s_d and thereafter following a greedy policy. Formally:

$$Q_d(s_d, X) \;=\; \mathbb{E}\Big[R_d(s_d, X) \;+\; \gamma \max_{X' \in \widehat{\mathcal{A}}_d(s'_d)} Q_d\big(s'_d,\, X'\big)\Big],$$

where s'_d is drone d's next local state after executing all actions in X. Note that the maximization is constrained to the masked action space $\widehat{\mathcal{A}}_d(s'_d)$, ensuring ethical compliance in future states as well. Standard temporal-difference updates can be applied (using replay buffers and target networks, if using deep Q-learning (85; 121)) as long as the *chosen CAS* X is always in $\widehat{\mathcal{A}}_d(s)$.

Algorithm 9 outlines the essential steps. Before selecting an action, drone d runs Algorithm 8 to obtain $\mathcal{F}_d$. The feasible status sets are then transformed into CASs forming the masked action space $\widehat{\mathcal{A}}_d(s)$.

Algorithm 9 Drone d: Ethics-Guided Q-Learning Algorithm

Input: (1) Q-network $Q_d(s, X)$ with parameters θ_d, (2) discount factor γ, (3) learning rate η, (4) exploration rate ϵ, (5) replay buffer $\mathcal{B}$, (6) FSS enumeration threshold τ, (7) norms $\mathcal{N}_d$, integrity constraints IC, action constraints AC.

for each episode or time step **do**
  Observe current state $s_d(t)$
  Compute feasible status sets: $\mathcal{F}_d \leftarrow$ Algorithm 8$\big(s_d(t), \mathcal{N}_d, IC, AC, \tau\big)$
  if $\mathcal{F}_d = \perp$ **then**
    Fallback: Execute no-op or safe maneuver; continue
  end if
  Masked actions: $\widehat{\mathcal{A}}_d(s_d(t)) \leftarrow \{X_{SS_d} \mid SS_d \in \mathcal{F}_d\}$
  With prob. ϵ: sample X uniformly from $\widehat{\mathcal{A}}_d(s_d(t))$
  Otherwise: $X = \arg\max_{X' \in \widehat{\mathcal{A}}_d(s_d(t))} Q_d(s_d(t), X')$
  Execute X, observe reward r_d (Section 5.4.3) and next state $s_d(t+1)$
  Store $(s_d(t), X, r_d, s_d(t+1))$ in $\mathcal{B}$
  Update Q: Sample minibatch from $\mathcal{B}$; for each (s, X, r, s'):

$$y = r + \gamma \max_{X' \in \widehat{\mathcal{A}}_d(s'_d)} Q_d(s'_d, X'; \theta_d),$$

$$L(\theta_d) = (y - Q_d(s, X; \theta_d))^2$$

$$\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} L(\theta_d)$$

end for

Because $\widehat{\mathcal{A}}_d(s)$ contains only ethically compliant actions, the drone never attempts forbidden maneuvers during training or execution.

QMIX for HQ Coordination. While individual drones learn local policies via Independent Q-Learning, the HQ learns to coordinate these drones to maximize team-level objectives. We adopt the QMIX (103), a widely used centralized training, decentralized execution algorithm.

Mixing Network. In QMIX, each drone d_i maintains a local Q-function Q_{d_i} (as in Appendix C.3.1), while the HQ learns a *mixing network*:

$$Q_{\text{tot}}(s_k^{\text{HQ}}, \boldsymbol{a}) \;=\; f\Big(Q_{d_1}(s_{d_1}, a_1),\; \ldots,\; Q_{d_m}(s_{d_m}, a_m);\; s_k^{\text{HQ}}\Big),$$

where $f(\cdot)$ is trained to approximate the *team-level Q-function*.

Algorithm 10 HQ k: QMIX Coordination Algorithm

Input: (1) Mixing network parameters θ, (2) discount factor γ, (3) learning rate η_{HQ}, (4) replay buffer $\mathcal{B}_{\text{HQ}}$, (5) monotonicity constraint on $f(\cdot)$.

for each episode or time step **do**
  Observe HQ state $s_k^{\text{HQ}}(t)$
  for each drone $d_i \in \mathcal{D}_k$ **do**
    Obtain local Q-values $Q_{d_i}(s_{d_i}(t), \cdot)$
  end for
  Compute $Q_{\text{tot}}(s_k^{\text{HQ}}(t), \boldsymbol{a}) = f(Q_{d_1}, \ldots, Q_{d_m}; s_k^{\text{HQ}})$
  $\boldsymbol{a}^{\text{HQ}}(t) = \arg\max_{\boldsymbol{a}} Q_{\text{tot}}(s_k^{\text{HQ}}(t), \boldsymbol{a})$
  HQ suggests $a_i^{\text{HQ}}(t)$ to each drone d_i
  Drone validation: Each d_i confirms or replaces $a_i^{\text{HQ}}(t)$ via FSS check
  Execute final joint action $\boldsymbol{a}(t)$
  Observe team reward $r_k(t)$ and next state $s_k^{\text{HQ}}(t+1)$
  Store $(s_k^{\text{HQ}}(t), \boldsymbol{a}(t), r_k(t), s_k^{\text{HQ}}(t+1))$ in $\mathcal{B}_{\text{HQ}}$
  Train mixing network:

$$y_{\text{tot}} = r_k + \gamma \max_{\boldsymbol{a}'} Q_{\text{tot}}(s_k^{\text{HQ}}(t+1), \boldsymbol{a}'; \theta^-)$$

$$L_{\text{HQ}}(\theta) = (y_{\text{tot}} - Q_{\text{tot}}(s_k^{\text{HQ}}(t), \boldsymbol{a}(t); \theta))^2$$

$$\theta \leftarrow \theta - \eta_{\text{HQ}} \nabla_\theta L_{\text{HQ}}(\theta)$$

end for

Motivating Example Setup. Consider step t in Figure 5.1:

(1) BLUE HQ aggregates state: observations from B1, B2, CCTV, grid values, etc.

(2) HQ queries local Q-values: Q_{B1} ranks "engage R1" highest; Q_{B2} ranks "move toward R2" highest.

(3) HQ computes Q_{tot} for joint actions. The joint action (B1 engages R1, B2 moves toward R2) yields the highest Q_{tot}.

(4) HQ sends suggestions to each drone.

(5) Each drone validates:

- B1: "engage R1" is in FSS-1 (feasible, since R1 is immediate threat). Accepted.
- B2: "move toward R2" is in FSS (movement permitted). Accepted.

(6) Actions execute: B1 fires at R1 (R1 eliminated), others move.

(7) Team reward computed: kill bonus for eliminating R1, survival bonuses, minus ammunition cost.

(8) HQ updates mixing network with this experience.

This completes one step of coordinated, ethically-compliant decision making. Overall, the design ensures *ethical compliance* is preserved. When the HQ selects drone actions, it tries to pick $(a_d)_{d\in\mathcal{D}_k}$ that jointly maximize Q_{tot}. However, each drone d *still* enforces its own feasibility mask. If the HQ suggests an infeasible CAS (e.g., a direct violation of deontic rules), the drone's CAS computation engine will reject it. Hence, the HQ cannot force a drone to violate DRs. Rather, it focuses on coordinating feasible CASs across the team to achieve higher-level goals. More details are shown in Appendix C.3.

5.4.5. RED: Non-Ethically-Guided Adversary

RED drones and HQ follow a standard MDP formulation structurally similar to BLUE. However, RED operates as a non-learning adversarial baseline rather than a co-learning adversary. Specifically, RED drones do not update policy parameters during training. Instead, RED drones select actions from their physically valid action set at each time step using either uniform random sampling or a hardcoded greedy policy that selects reward-maximizing actions from the valid set (see Appendix C.2.7 for reward definitions). Unlike BLUE, RED's action space is not contrained through ethical norms. This design models an adversary operating without ethical or legal restrictions, i.e., enabling destruction of city infrastructure and direct engagement of BLUE drones. The RED team also operates with perfect communication, representing a worst-case adversarial assumption. This asymmetry models adversaries who operate without legal or ethical restrictions while providing a consistent and reproducible baseline for evaluating BLUE's norm-compliant policies.

5.5. Experimental Assessment

Our experiments are designed to assess the effectiveness of GUARDIAN, including the cost of compliance with ethical/legal norms expressed in deontic logic. The detailed assumptions of the GUARDIAN testbed are in Appendix C.4.

5.5.1. Setting

We simulated an urban environment via a 2-d $N \times N$ grid with $N \in \{64, 128\}$. Each cell was assigned an initial value $v_{i,j}(0)$, picked uniformly at random from $[0, 100]$, representing the importance of city locations. All drones had initial battery capacity $B_d(0) = 100$ (depleted at 0.5 units per time step), view range $r_d = 5$, fire range $f_d = 1$, and initial payload $p_d(0) = 3$. $|\mathcal{C}_1| = 3$ CCTV cameras, with view range $r_c = 10$, were placed at random at the start of each episode, and their positions then remained fixed for the duration of that episode. The drones were trained using Deep Independent Q-Learning (121). Each BLUE (resp., RED) drone maintained separate Q-value functions for the objectives of protecting (resp., damaging) the city and ethical compliance (BLUE only). The HQs (details in Appendix C.1.10) were trained using QMIX to learn coordinated strategies that maximize team-level objectives. They utilize the individual Q-values of their drones to compute a total Q-value for joint action selection.

The transition function $P_d(s, X, s')$ is deterministic: movement succeeds with probability 1 if the target cell is valid and unoccupied, and firing actions succeed with probability 1 if preconditions are satisfied. This deterministic setting isolates the effects of deontic compliance from action uncertainty, following standard practice in multi-agent RL benchmarks (107; 94). The only stochastic element is communication, modeled as a Bernoulli random variable $C_d(t) \sim \text{Bernoulli}(p_{\text{comm}})$ with $p_{\text{comm}} = 0.8$ for BLUE drones and $p_{\text{comm}} = 1.0$ for RED drones. The 80% reliability for BLUE reflects realistic battlefield communication conditions under potential jamming (41), while RED's perfect communication represents a worst-case adversarial assumption. Details are provided in Appendix C.1.12.

We varied the number $|\mathcal{D}_1|$ of BLUE drones in $\{16, 32, 64\}$ and the ratio of BLUE to RED drones (*B:R ratio*) (i.e., $\frac{|\mathcal{D}_1|}{|\mathcal{D}_2|}$) in $\{1{:}1, 2{:}1, 3{:}1, 1{:}2, 1{:}3\}$. *In some situations, the defender may have more drones than the attacker (so the B:R ratio would exceed 1), but in other situations, the B:R ratio may be either above or below 1.* Let $N_{\text{ep}} = 5{,}000$ denote the number of training episodes. Each experiment was evaluated over N_{ep} episodes.

5.5.2. Performance Metrics

We measured the main performance metrics outlined below (additional metrics are in Appendix C.5).

- *Reward.* Cumulative reward accrued by BLUE drones during an episode.
- *City Protection.* Quantifies the effectiveness of BLUE in protecting the city. We start by computing a "raw" protection value $P_{\text{raw}} = \frac{\sum_{(i,j)\in N\times N} \max(0, v_{i,j}(final))}{\sum_{(i,j)\in N\times N} v_{i,j}(0)}$. We then compute the theoretical minimum protection $L = 1 - \frac{D_{\max}}{\sum_{(i,j)\in N\times N} v_{i,j}(0)}$ achievable under the assumption that the RED team acts optimally (with perfect coordination and no interference from the BLUE team). In the formula, $D_{\max}$ is the sum of the top-P_{red} cell values and P_{red} is the total payload of the RED drones. The final city protection value is 1 if $P_{\text{raw}} \geq 1$, $\frac{P_{\text{raw}} - L}{1 - L}$ if $L < P_{\text{raw}} < 1$, and 0 if $P_{\text{raw}} \leq L$.
- *Win Rate.* Percentage of scenarios where BLUE neutralized all RED drones before the scenario timed out.
- *Threat Neutralization Steps.* Number of steps taken to eliminate all enemy threats.

5.5.3. Results

Reward values are shown in Figure 5.3. We made three observations.

(1) (Obs 1) As expected, compliance with DRs generally worsens performance.

(2) (Obs 2) Interestingly, on the 64x64 grid with 32 and 64 BLUE drones, compliance with DRs appears to improve performance when RED drones are the majority (1:2 and 1:3 ratios).

(3) (Obs 3) When the problem is larger (i.e., the grid size is larger or the number of drones is larger), the reward when complying with DRs is closer to and then larger than the value without DRs.

A possible explanation for (Obs 2,3) is that the deontic ethical/legal norms limit the behavior of the BLUE drones, making the decision space smaller, compared to RED. When the problem of learning the optimal strategy is relatively large for RED, these deontic rules help the BLUE drones learn a strategy by reducing the decision space.

For the other performance metrics, in order to capture how compliance with DRs affects performance, given a *B:R* drone ratio and a performance metric M such as those defined above, we defined a *compliance cost*

$$CC(B{:}R, M) = \frac{1}{N_{\text{ep}}} \sum_{e=1}^{N_{\text{ep}}} CC(B{:}R, M, e),$$

where $CC(B{:}R, M, e)$ is the value of M in the compliance case (i.e., when the BLUE drones comply with ethical norms) divided by the value of M in the no compliance case, at episode e. The division by N_{ep} averages the compliance cost over all episodes. Table 5.1

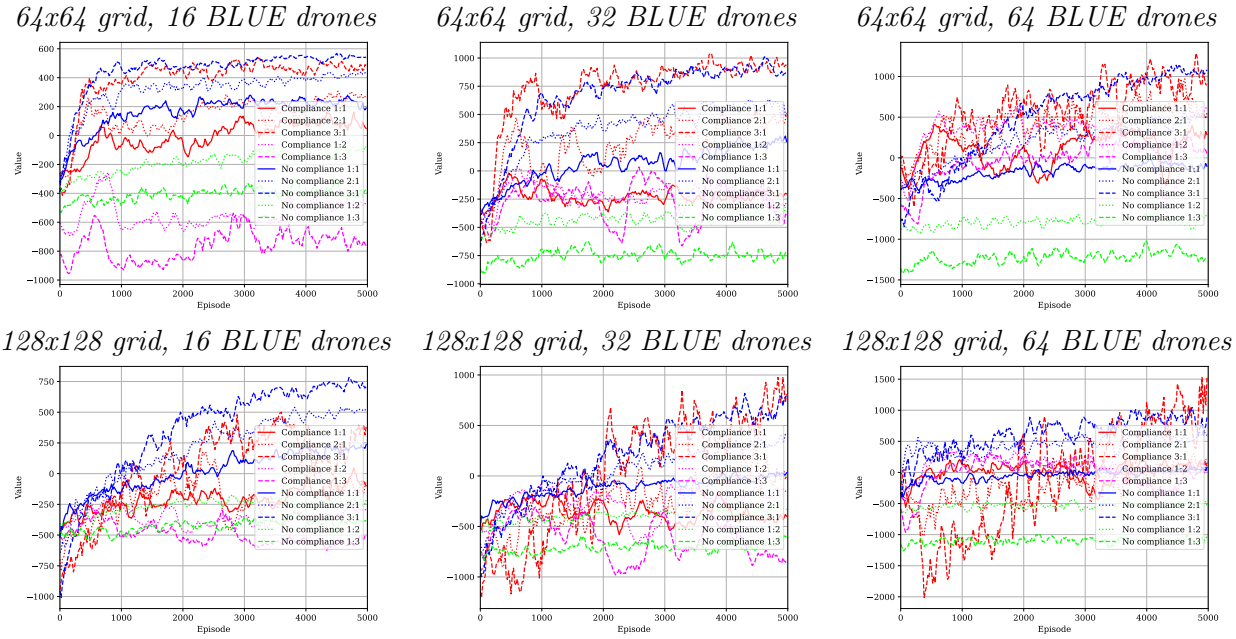


Figure 5.3. Reward to BLUE.

shows compliance costs for all $B{:}R$ ratios when varying grid size and number of BLUE drones. Detailed graphs similar to Figure 5.3 are shown in the Appendix.

City protection. Compliance with DRs appears to improve BLUE's performance. (Obs 4) Specifically compliance with DRs causes worse city protection in just 4 cases (up to 14%), whereas in all other cases it improves protection (up to 20.9%).

Win rates show an interesting trend. (Obs 5) With 16 BLUE drones, compliance with DRs worsens performance (up to 43.8%), but in the case of 32 or 64 BLUE drones, performance appears to improve noticeably (up to 9.576 times). Again, as with (Obs 2,3) this is likely because of the reduced decision space for BLUE. (Obs 6) In addition, when RED drones are the majority, the BLUE team only wins in one case which should not be a huge surprise.

City protection (higher is better)

	16 BLUE drones		32 BLUE drones		64 BLUE drones	
	64x64	**128x128**	**64x64**	**128x128**	**64x64**	**128x128**
1:1	0.860	1.025	0.967	1.044	1.156	1.165
2:1	0.962	1.012	1.082	1.196	1.209	1.283
3:1	0.980	1.083	1.152	1.214	1.187	1.313
1:2	1.041	1.025	1.076	1.092	1.120	1.106
1:3	1.109	1.009	1.076	1.032	1.076	1.093

Win rate (higher is better)

	16 BLUE drones		32 BLUE drones		64 BLUE drones	
	64x64	**128x128**	**64x64**	**128x128**	**64x64**	**128x128**
1:1	0.699	0.661	1.224	2.779	8.538	–
2:1	0.912	0.978	1.183	4.726	5.694	9.576
3:1	0.930	0.989	1.123	3.897	2.103	6.911
1:2	0.562	–	–	–	–	–
1:3	–	–	–	–	–	–

Threat neutralization steps (lower is better)

	16 BLUE drones		32 BLUE drones		64 BLUE drones	
	64x64	**128x128**	**64x64**	**128x128**	**64x64**	**128x128**
1:1	2.077	1.137	1.101	0.995	0.937	1.000
2:1	1.540	1.110	0.850	0.850	0.679	0.916
3:1	1.073	0.912	0.588	0.722	0.658	0.729
1:2	1.013	1.000	1.000	1.000	1.000	1.000
1:3	1.000	1.000	1.000	1.000	1.000	1.000

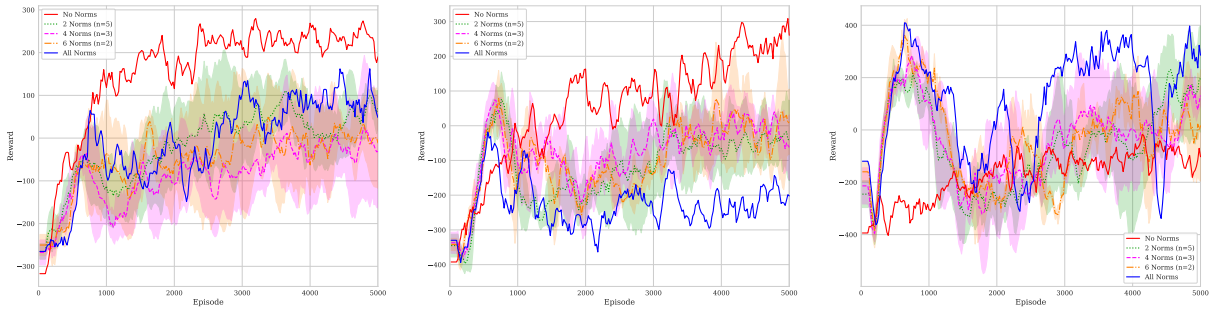
Table 5.1. Compliance cost when varying *B:R* ratio, grid size, and number of BLUE drones. Note that "–" means that BLUE did not win.

Threat neutralization steps. (Obs 7) As expected, in some cases, compliance with DRs increases the number of threat neutralization steps. Still, in most cases, compliance yields better performance.

Appendix C.6 reports charts with results obtained for each metric, *B:R* ratio, grid size, and number of BLUE drones.

5.5.4. Impact of Norm Combinations

The experiments above use either the complete set of eight norms or no norms. To understand how specific norm subsets contribute to system performance, we conducted



(a) 64×64 grid, 16 drones (1:1 ratio) (b) 64×64 grid, 32 drones (1:1 ratio) (c) 64×64 grid, 64 drones (1:1 ratio)

Figure 5.4. Test rewards across norm combinations in symmetric (1:1) competitive scenarios. Lines represent mean performance with standard deviation bands.

additional experiments with norm sets of varying sizes: $|\mathcal{N}| \in \{0, 2, 4, 6, 8\}$. Each combination was strategically selected to test specific hypotheses about norm interactions and their operational impact. The experiments maintain a 1:1 BLUE-to-RED drone ratio across all configurations for symmetric competitive scenario.

Figure 5.4 illustrates the results. Three patterns emerge (detailed in Appendix C.7): (Obs 8) In sparse deployments (16 drones), no-norm configuration achieves highest rewards. (Obs 9) Partial norm sets consistently underperform both extremes. (Obs 10) As drone density increases, complete norm sets achieve superior performance.

The transition occurs at approximately 32 drones. These results demonstrate that norm completeness matters more than norm count. Hence, partial ethical frameworks may perform worse than either complete frameworks or unconstrained operation.

5.5.5. Computation Time

We now report the runtime for some experiments which were conducted on a workstation with an Intel® Core™ i9-10980XE CPU (18 cores, 36 threads) running at 3.00GHz with

251GB RAM. Most tasks were performed on the CPU. Neural network training with QMIX used the NVIDIA RTX A6000 GPU.

Training for 5,000 episodes with 64 blue drones on a 64×64 grid required about 630 (resp. 80) hours with (resp. without) DRs. This reflects the expected computational overhead of reasoning with deontic constraints during the learning process.

Once trained offline, GUARDIAN is extremely practical during operations. Per-step decision times for 25 episodes with a maximum of 200 steps per episode for each configuration, on a 64x64 grid with 16, 32, and 64 drones and 1:1 drone ratio in milliseconds are shown in Table 5.2.

BLUE drones	Compliance	CAS computation	QMIX
16	**Yes**	215.576 ± 54.560	23.688 ± 42.256
	No	0	15.344 ± 63.888
32	**Yes**	446.760 ± 131.920	26.216 ± 50.960
	No	0	23.568 ± 63.848
64	**Yes**	554.856 ± 92.096	70.832 ± 31.520
	No	0	41.616 ± 51.112

Table 5.2. Per-step decision time (milliseconds) on a 64x64 grid with 1:1 drone ratio.

When increasing the drone count from 16 to 64, GUARDIAN's CAS computation time grows by a factor of 2.6 (from 215.6ms to 554.9ms) and the neural network inference time (QMIX) increases by a factor of 3 (from 23.7ms to 70.8ms)—both are less than the 4x increase in drone count. Despite the additional computational requirements, the total decision time with compliance remains well within acceptable bounds for real-time autonomous control—even with 64 drones, the average decision time is 625.7ms, showing GUARDIAN's practical utility in real-world scenarios.

5.6. Limitations and Future Work

GUARDIAN operates on a 2D grid where drones move in four cardinal directions, and the reported experiments use deterministic action outcomes with success probability 1.0. While the testbed supports probabilistic transitions via uniform and normal distributions, we have not evaluated GUARDIAN under stochastic action outcomes for operational simplicity. Also, extending the framework to 3D environments with altitude variations and altitude-dependent regulations, as well as assessing robustness under action uncertainty, remain directions for future work. However, both cases will be computationally expensive.

Our current formulation models RED drones as a non-learning baseline. Extending GUARDIAN to incorporate a co-learning adversary via self-play or minimax formulations would enable analysis of how adversaries adapt their strategies over time based on observed BLUE behavior, and how BLUE drones can develop countermeasures while maintaining ethical compliance. We leave this adversarial co-evolution analysis to future work.

Current experiments assume homogeneous drone capabilities within each team. In practice, defender and attacker fleets may comprise heterogeneous drones with varying payload capacities, fire ranges, battery endurance, and maneuverability. The BLUE-to-RED ratio in current experiments could serve as a proxy for capability asymmetries, but explicit heterogeneous modeling would provide more realistic assessments.

While GUARDIAN ensures legal compliance through deontic logic constraints, the learned policies remain black-box neural networks. Future work should investigate explainability methods that allow human operators to understand why specific actions were selected and how norms influenced decisions. Similarly, models trained on 64×64 grids cannot be directly applied to 128×128 or larger environments without retraining. Hence,

another future work can explore transfer learning mechanisms that enable models trained on smaller grids to generalize to larger or differently configured environments, which would reduce the computational burden of deployment in new cities.

5.7. Conclusions

Defense officials have been worried about the impact of ethical/legal norms on the behavior of autonomous drones. Over the years, there has been concern that this ties their hands, while the enemy's hands are free. We describe GUARDIAN, a testbed for experimenting with different ethical/legal norms in the context of autonomous, RL-based drone swarms. Designed using input from security experts in the US, Netherlands, India, and Israel, GUARDIAN expresses ethical/legal norms in deontic logic (44; 45; 53), and combines it with RL.

Because ethical/legal norms vary from country to country and situation to situation, the inputs we got from security experts may not be comprehensive. But they suffice to demonstrate that GUARDIAN supports testing the impact of ethical/legal norms in different settings. We present results showing that the imposition of legal/ethical norms may not negatively tie the hands of the defender. Instead, when RED has more drones, the deontic compliance rules may help the defender focus and be more efficient in finding policies that both satisfy the required norms, and yield better reward.

CHAPTER 6

Future Directions and Conclusion

The components developed in this dissertation have been independently validated and demonstrate the feasibility of responsible urban drone defense. The natural progression of this research involves comprehensive integrated evaluation where these components operate together within realistic operational scenarios. To initiate that frontier, we have developed DUCK (Drone Urban Cyberdefense) (35), a high-fidelity 3D simulation testbed described in this chapter. DUCK provides infrastructure for systematic evaluation of multi-agent defensive strategies, exploration of emergent behaviors when multiple components operate together, and assessment of integrated system performance under diverse operational conditions. DUCK allows defenders to use CCTVs, Blue drones, and cyber attacks to defend against swarms of Red drones. Defenders can simulate attacks and assess their impact.

6.1. DUCK Implementation

The DUCK architecture (Figure 6.1) comprises over 10,100 lines of code in C++ and Python. DUCK uses a customized 3D model based on the Unreal Engine and Microsoft's AirSim (109) simulator to visualize multiple drones in 3D environments, navigation, and real-time drone state management. We extended AirSim to include CCTVs, drone-drone attacks, destruction of regions, hacking, battery and payload constraints, and built high-level Python APIs for these features. Key visualization, control, communications, and

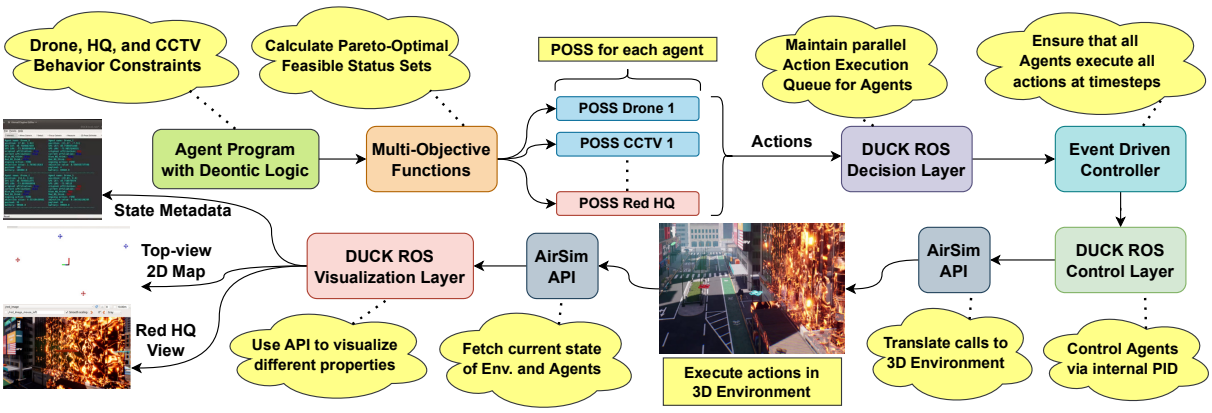


Figure 6.1. Simplified DUCK Testbed Architecture for a single execution cycle (timestep).

concurrency aspects were built within the Robot Operating System (ROS) connected to AirSim.

The ROS component implements DUCK's decision, control, and visualization layers. Each layer has separate nodes running in parallel for message communications. The decision layer identifies actions each agent decides to perform at time t by finding a Pareto-optimal set of actions compatible with that agent's program. Not all attempted actions succeed. The control layer injects stochasticity into agent actions and determines which attempted actions succeed. For example, a blue drone bd may fire at red drone rd, but ROS may determine that rd is not destroyed. The visualization layer displays ground truth, attempted actions, objective values, camera images from agents, and real-time movement on maps.

6.2. DUCK Capabilities

The DUCK demo[1] allows a user to set the number and capabilities (e.g. payload, firing range, battery) of the blue and red drones, and the number of CCTVs. Figure 6.2

[1]Available at: https://sites.northwestern.edu/nsail/projects/duck

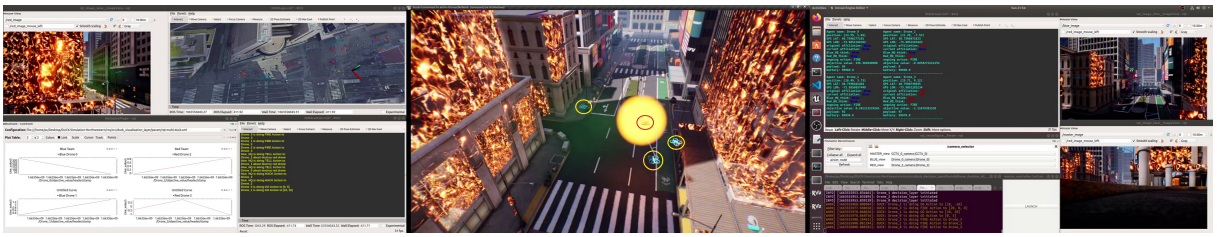


Figure 6.2. DUCK 3-Screen Demonstration. The middle screen visualizes GT. Left and right screens show other technical details.

shows the demonstration on 3 screens shortly after launching. The demonstration allows users to configure the number and capabilities of blue and red drones (payload, firing range, battery) and the number of CCTVs. The system provides multiple synchronized visualization screens. One screen shows Red HQ's view of drone cameras and displays how objective function values change as the simulation proceeds. Another screen shows ground truth in the 3D environment, depicting multi-drone engagements. A third screen shows Blue HQ's view of drone cameras and CCTV feeds, and visualizes drone state information including GPS coordinates, battery, payload, hack status, and operational status. The interface enables event-driven simulation where users can step through discrete timesteps.

DUCK allows defenders to simulate attacks on urban regions and assess efficacy of diverse defenses including drone deployments and cyber attacks, thereby computing how to minimize damage under cost constraints. The testbed supports systematic evaluation of defensive strategies across varying team compositions, operational constraints, and adversarial tactics. It can generate photorealistic data for sim2real training (37; 100; 99).

6.3. Limitations and Future Directions

While this dissertation advances autonomous, compliant urban drone defense, several limitations remain.

Transfer Learning Across Environments. GUARDIAN models trained on 64×64 grids cannot be applied to 128×128 or larger environments without retraining. While feasible status set computation remains valid regardless of grid size, the Q-learning components require retraining for different environments. Developing transfer learning mechanisms that enable models trained on smaller grids to generalize to larger configurations would reduce the computational burden of deployment.

Scalability via Graph Coarsening. Prior work on Stackelberg security games for drone defense (86) introduced delta-coarsening as a scalability mechanism. Given a large city graph, the algorithm compresses the city into neighborhoods, solves sub-games within each neighborhood, and expands solutions back to the original scale. Integrating delta-coarsening with GUARDIAN could enable deployment to cities of arbitrary size: partition a large city into neighborhoods, train policies for each neighborhood configuration, coordinate neighborhood-level defenses through hierarchical command, and use the coarsening-expansion process to map policies to full-scale operations.

Geographic Scope of STATE. STATE has been evaluated only in The Hague with annotations from Dutch police experts. Extension to multiple cities with different urban layouts, regulatory environments, and cultural contexts is necessary to establish generalizability.

Intent-Based Classification. Current threat classification of DEWS operates as binary (threatening/non-threatening) without modeling attacker intent. The Dutch police experts provided intent annotations explaining why trajectories were classified as threatening, but this information has not been incorporated into classification models. Leveraging

intent data could enable threat assessment that distinguishes reconnaissance, surveillance, delivery, and attack trajectories.

Heterogeneous Drone Capabilities. GUARDIAN experiments assume homogeneous drone capabilities within each team. In practice, fleets may comprise drones with varying payload capacities, fire ranges, battery endurance, and maneuverability. The Blue-to-Red ratio could serve as a proxy for capability asymmetries, but explicit heterogeneous modeling would provide more realistic assessments.

Model Interpretability. While GUARDIAN ensures legal compliance through deontic logic constraints, the learned policies remain black-box neural networks. The responsibility framework addresses legal and ethical compliance with governance requirements, but does not provide explanations for specific decisions. Future work should investigate explainability methods that allow operators to understand why specific actions were selected and how norms influenced decisions.

Training Overhead. Training with deontic compliance in GUARDIAN requires approximately 630 hours compared to 80 hours without compliance for equivalent configurations. While inference times remain practical (under 700ms per decision), the training overhead presents barriers to rapid iteration and deployment in new environments.

Dual-Use Considerations. The STATE model for generating threatening trajectories presents dual-use concerns: the capability that enables defenders to anticipate attacks could assist adversaries in planning trajectories. State actors with significant computational resources pose different considerations than non-state actors with limited capabilities. Responsible deployment requires access controls and monitoring of model usage.

6.4. Conclusion

This dissertation has addressed the critical challenge of enabling proactive, legally compliant defense of populated regions from hostile drone activities. The work has developed four complementary frameworks: DEWS for early threat prediction from minimal trajectory observations, STATE for generating synthetic training data addressing data scarcity, POSS for legally compliant multi-objective decision-making, and GUARDIAN for reinforcement learning under hard constraints. Each contribution has been independently validated, demonstrating that responsible urban drone defense requires simultaneous attention to technical effectiveness, legal compliance, and empirical rigor. The ongoing DUCK testbed provides infrastructure for future integrated evaluation of these components within realistic operational scenarios.

The path forward involves transitioning from component-level validation to comprehensive integration, from simplified testbed environments to high-fidelity simulation, and ultimately to operational deployment. The technical challenges of achieving robustness against sophisticated adversaries and generalizing across diverse operational contexts remain substantial. However, this research establishes that threats can be identified early enough to enable meaningful response, data scarcity can be addressed through principled synthetic generation, legal constraints can be formalized and integrated into automated reasoning, and adaptive learning can proceed under normative boundaries. The vision of defensive systems that are simultaneously effective, compliant, adaptive, and transparent is achievable through rigorous interdisciplinary inquiry, and this dissertation represents meaningful progress toward that goal.

References

[1] Ajith, V., and Jolly, K. Unmanned aerial systems in search and rescue applications with their path planning: a review. In *Journal of Physics: Conference Series* (2021), vol. 2115, IOP Publishing, p. 012020.

[2] Alferes, J. J., Banti, F., and Brogi, A. An event-condition-action logic programming language. In *JELIA Conference* (2006).

[3] Almohammad, A., and Speckhard, A. Isis drones: Evolution, leadership, bases, operations and logistics. *The International Center for the Study of Violent Extremism 5* (2017).

[4] Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U. Safe reinforcement learning via shielding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence* (2018), pp. 2669–2678.

[5] Balkan, S. A global battlefield? rising drone capabilities of non-state armed groups and terrorist organizations. Tech. rep., 2019.

[6] Banerjee, I., Honnappa, H., and Rao, V. Off-line estimation of controlled markov chains: Minimaxity and sample complexity. *Operations Research* (2025).

[7] Banerjee, I., Rao, V., and Honnappa, H. Adaptive estimation of the transition density of controlled markov chains. *arXiv preprint arXiv:2505.14458* (2025).

[8] Barua, L., Zou, B., and Zhou, Y. Machine learning for international freight transportation management: A comprehensive review. *Research in Transportation Business & Management 34* (2020), 100453.

[9] Becker, S., Hug, R., Hübner, W., Arens, M., and Morris, B. T. Generating synthetic training data for deep learning-based UAV trajectory prediction. *CoRR abs/2107.00422* (2021).

[10] Bell, C., Nerode, A., Ng, R. T., and Subrahmanian, V. Implementing deductive databases by linear programming. In *PODS* (1992).

[11] Bell, C., Nerode, A., Ng, R. T., and Subrahmanian, V. Mixed integer programming methods for computing nonmonotonic deductive databases. *J. ACM 41*, 6 (1994), 1178–1215.

[12] Bertsekas, D. *Reinforcement learning and optimal control*, vol. 1. Athena Scientific, 2019.

[13] Best, K. L., Schmid, J., Tierney, S., Awan, J., Beyene, N. M., Holliday, M. A., Khan, R., and Lee, K. *How to Analyze the Cyber Threat from Drones: Background, Analysis Frameworks, and Analysis Tools.* RAND Corporation, Santa

Monica, CA, 2020.

[14] Bharilya, V., and Kumar, N. Machine learning for autonomous vehicle's trajectory prediction: A comprehensive survey, challenges, and future research directions. *Vehicular Communications* (2024), 100733.

[15] Bloom, D. C-uas in urban environments: Challenges and opportunities. D-Fend Solutions Blog (March 20, 2024), 2024. Describes dense urban terrain providing cover and complicating counter-drone operations.

[16] Bolonkin, M., Chakrabarty, S., Molinaro, C., and Subrahmanian, V. Judicial support tool: Finding the k most likely judicial worlds. In *International Conference on Scalable Uncertainty Management* (2024), Springer, pp. 53–69.

[17] Bonatti, R., Wang, W., Ho, C., Ahuja, A., Gschwindt, M., Camci, E., Kayacan, E., Choudhury, S., and Scherer, S. Autonomous aerial cinematography in unstructured environments with learned artistic decision-making. *Journal of Field Robotics 37*, 4 (2020), 606–641.

[18] Boukoberine, M. N., Zhou, Z., and Benbouzid, M. A critical review on unmanned aerial vehicles power supply and energy management: Solutions, strategies, and prospects. *Applied Energy 255* (2019), 113823.

[19] Brandes, U. A faster algorithm for betweenness centrality. *Journal of mathematical sociology 25*, 2 (2001), 163–177.

[20] Brinkmeier, M. Pagerank revisited. *ACM Trans. Internet Techn. 6*, 3 (2006), 282–301.

[21] Cai, Y., Zhang, H., Su, H., Zhang, J., and He, Q. The bipartite edge-based event-triggered output tracking of heterogeneous linear multiagent systems. *IEEE Trans. Cybern. 53*, 2 (2023), 967–978.

[22] Calegari, R., Ciatto, G., Mascardi, V., and Omicini, A. Logic-based technologies for multi-agent systems: Summary of a systematic literature review. In *AAMAS Conference* (2021).

[23] Chandru, V., and Hooker, J. N. Extended Horn Sets In Propositional Logic. *J. ACM 38*, 1 (1991), 205–221.

[24] Cheikh, M., Jarboui, B., Loukil, T., and Siarry, P. A method for selecting pareto optimal solutions in multiobjective optimization. *Journal of Informatics and mathematical sciences 2*, 1 (2010), 51–62.

[25] Chen, F., Wang, X., Zhao, Y., Lv, S., and Niu, X. Visual object tracking: A survey. *Computer Vision and Image Understanding 222* (2022), 103508.

[26] Chen, J., and Sayed, A. H. Distributed pareto optimization via diffusion strategies. *IEEE J. Sel. Top. Signal Process. 7*, 2 (2013), 205–220.

[27] Chen, X., Du, Y., Xia, L., and Wang, J. Reinforcement recommendation with user multi-aspect preference. In *WWW Conference* (2021).

[28] Clocksin, W. F., and Mellish, C. S. *Programming in PROLOG*. Springer Science & Business Media, 2003.

[29] Coello, C. A. C. *Evolutionary algorithms for solving multi-objective problems*.

Springer, 2007.

[30] Costantini, S., and Tocchio, A. A logic programming language for multi-agent systems. In *JELIA Conference* (2002).

[31] De Cubber, G. Explosive drones: How to deal with this new threat? In *Proceedings of the 2019 International Workshop on Countering Unmanned Aerial Systems* (Brussels, Belgium, 2019).

[32] de Wit, V., Doder, D., Meyer, J. J., et al. Probabilistic deontic logics for reasoning about uncertain norms. *Journal of Applied Logics 2631*, 2 (2023), 193.

[33] Deb, K., and Jain, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE Trans. Evol. Comput. 18*, 4 (2014), 577–601.

[34] Deb, T., de Laaf, S., La Gatta, V., Lemmens, O., Lindelauf, R., van Meerten, M., Meerveld, H., Neeleman, A., Postiglione, M., and Subrahmanian, V. A drone early warning system (dews) for predicting threatening trajectories. *IEEE Intelligent Systems* (2025), 1–10.

[35] Deb, T., Dix, J., Jeong, M., Molinaro, C., Pugliese, A., Li, A. Q., Santos Jr, E., Subrahmanian, V., Yang, S., and Zhang, Y. Duck: A drone-urban cyber-defense framework based on pareto-optimal deontic logic agents. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2023), vol. 37, pp. 16425–16427.

[36] Deb, T., Jeong, M., Molinaro, C., Pugliese, A., Li, A. Q., Santos, E., Subrahmanian, V., and Zhang, Y. Declarative logic-based pareto-optimal agent decision making. *IEEE Transactions on Cybernetics* (2024).

[37] Deb, T., Rahmun, M., Bijoy, S. A., Raha, M. H., and Khan, M. A. Uucthymp: Towards tracking dispersed crowd groups from uavs. In *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), IEEE, pp. 1–8.

[38] Dhariwal, P., and Nichol, A. Diffusion models beat gans on image synthesis. In *Advances in Neural Information Processing Systems* (2021), vol. 34, pp. 8780–8794.

[39] Dikmen, M., and Burns, C. M. Autonomous driving in the real world: Experiences with tesla autopilot and summon. In *AutomotiveUI Conference* (2016).

[40] Drew, T., and Gini, M. Implantable medical devices as agents and part of multiagent systems. In *AAMAS Conference* (2006).

[41] Du, Z., Wu, C., Yoshinaga, T., Yau, K.-L. A., Ji, Y., and Li, J. Machine learning for large-scale optimization in 6g wireless networks. *IEICE Transactions on Communications 103*, 6 (2020), 678–689.

[42] Egeland, K. Lethal autonomous weapon systems under international humanitarian law. *Nordic Journal of International Law 85*, 2 (2016), 89–118.

[43] Ehrgott, M. *Multicriteria Optimization*. Springer, 2005.

[44] Eiter, T., Subrahmanian, V., and Pick, G. Heterogeneous active agents, i: Semantics. *Artificial Intelligence 108*, 1-2 (1999), 179–255.

[45] Eiter, T., Subrahmanian, V., and Rogers, T. J. Heterogeneous active agents,

iii: Polynomially implementable agents. *Artificial Intelligence 117*, 1 (2000), 107–167.

[46] European Union Aviation Safety Agency. Drone incident management at aerodromes. Tech. rep., EASA, Cologne, Germany, 2021. Noting that a consumer drone can penetrate 5 km of protected airspace in under 4 minutes.

[47] Federal Aviation Administration. Operation and certification of small unmanned aircraft systems (part 107). Federal Register Rule 81 FR 42063, 2016. U.S. domestic drone regulation outlining operational restrictions and requirements.

[48] Føllesdal, D., and Hilpinen, R. Deontic logic: An introduction. In *Deontic logic: Introductory and systematic readings*, vol. 33. Springer, 1971, pp. 1–35.

[49] Fotouhi, A., Qiang, H., Ding, M., Hassan, M., Giordano, L., Garcia-Rodriguez, A., and Yuan, J. Survey on uav cellular communications: Practical aspects, standardization advancements, regulation, and security challenges. *IEEE Communications Surveys & Tutorials 21*, 4 (2019), 3417–3442.

[50] Gabbay, D. M., Horty, J. F., Parent, X., van der Meyden, R., and van der Torre, L. *The Handbook of Deontic Logic and Normative Systems*. College Publications, 2021.

[51] Gambs, S., Killijian, M.-O., and del Prado Cortez, M. N. Next place prediction using mobility markov chains. In *Proceedings of the first workshop on measurement, privacy, and mobility* (2012), pp. 1–6.

[52] Georgiou, H., Karagiorgou, S., Kontoulis, Y., Pelekis, N., Petrou, P., Scarlatti, D., and Theodoridis, Y. Moving objects analytics: Survey on future location & trajectory prediction methods. *arXiv preprint arXiv:1807.04639* (2018).

[53] Governatori, G., Rotolo, A., Sartor, G., Gabbay, D., Horty, J., Parent, X., van der Meyden, R., and van der Torre, L. Logic and the law: philosophical foundations, deontics, and defeasible reasoning. *Handbook of Deontic Logic and Normative Reasoning 2* (2021), 655–760.

[54] Hao, Y., Liu, L., and Feng, G. Event-triggered cooperative output regulation of heterogeneous multiagent systems under switching directed topologies. *IEEE Trans. Cybern. 53*, 2 (2023), 1026–1038.

[55] Hjelmblom, M., and Odelstad, J. jDALMAS: A java/prolog framework for deontic action-logic multi-agent systems. In *KES-AMSTA Symposium* (2009).

[56] Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems* (2020), vol. 33, pp. 6840–6851.

[57] Hoenig, M. Hezbollah and the use of drones as a weapon of terrorism. *Public Interest Report 67*, 2 (2014).

[58] Huang, D., Song, X., Fan, Z., Jiang, R., Shibasaki, R., Zhang, Y., Wang, H., and Kato, Y. A variational autoencoder based generative model of urban human mobility. In *2019 IEEE conference on multimedia information processing and retrieval (MIPR)* (2019), IEEE, pp. 425–430.

[59] Hughes, M., and Hess, J. An assessment of lone wolves using explosive-laden consumer drones in the united states. *Global Security and Intelligence Studies 2*, 1 (2016), 62–84.

[60] International Institute of Humanitarian Law, Ed. *The San Remo Handbook on Rules of Engagement*. IIHL, Sanremo, Italy, 2009. Establishes that ROE must align with international law and domestic legal parameters.

[61] Jin, K., Han, S., Baek, D., and Lee, H. L. Small drone detection using hybrid beamforming 24 ghz fully integrated cmos radar. *Drones 9*, 7 (2025), 453.

[62] Julka, S., Sowrirajan, V., Schloetterer, J., and Granitzer, M. Conditional generative adversarial networks for speed control in trajectory simulation. In *Machine Learning, Optimization, and Data Science* (Cham, 2022), G. Nicosia, V. Ojha, E. La Malfa, G. La Malfa, G. Jansen, P. M. Pardalos, G. Giuffrida, and R. Umeton, Eds., Springer International Publishing, pp. 436–450.

[63] Kahagalage, S. D., Turan, H. H., Jalalvand, F., and Sawah, S. E. A novel graph-theoretical clustering approach to find a reduced set with extreme solutions of pareto optimal solutions for multi-objective optimization problems. *J. Glob. Optim. 86*, 2 (2023), 467–494.

[64] Kim, J.-H., and Kum, D.-S. Threat prediction algorithm based on local path candidates and surrounding vehicle trajectory predictions for automated driving vehicles. In *2015 IEEE Intelligent Vehicles Symposium (IV)* (2015), IEEE, pp. 1220–1225.

[65] Kulkarni, V., Moro, A., and Garbinato, B. Mobidict: A mobility prediction system leveraging realtime location data streams. In *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming* (2016), pp. 1–10.

[66] Lee, D., Kim, H., Choi, Y., and Kim, J. Development of autonomous operation agent for normal and emergency situations in nuclear power plants. In *ICSRS Conference* (2021).

[67] Liang, J. J., Qiao, K., Yu, K., Qu, B., Yue, C., Guo, W., and Wang, L. Utilizing the relationship between unconstrained and constrained pareto fronts for constrained multiobjective optimization. *IEEE Trans. Cybern. 53*, 6 (2023), 3873–3886.

[68] Lin, L., Li, W., Bi, H., and Qin, L. Vehicle trajectory prediction using lstms with spatial–temporal attention mechanisms. *IEEE Intelligent Transportation Systems Magazine 14*, 2 (2021), 197–208.

[69] Lindahl, L., and Odelstad, J. Normative positions within an algebraic approach to normative systems. *Journal of Applied Logic 2*, 1 (2004), 63–91.

[70] Liu, Y., Zhu, N., and Li, M. Solving many-objective optimization problems by a pareto-based evolutionary algorithm with preprocessing and a penalty mechanism. *IEEE Trans. Cybern. 51*, 11 (2021), 5585–5594.

[71] Liu, Z., An, P., Yang, Y., Qiu, S., Liu, Q., and Xu, X. Vision-based drone detection in complex environments: A survey. *Drones 8*, 11 (2024), 643.

[72] Lloyd, J. W. *Foundations of logic programming*. Springer Science & Business Media, 2012.

[73] Luca, M., Barlacchi, G., Lepri, B., and Pappalardo, L. A survey on deep learning for human mobility. *ACM Computing Surveys (CSUR) 55*, 1 (2021), 1–44.

[74] Ma, L., Huang, M., Yang, S., Wang, R., and Wang, X. An adaptive localized decision variable analysis approach to large-scale multiobjective and many-objective optimization. *IEEE Trans. Cybern. 52*, 7 (2022), 6684–6696.

[75] Ma, L., Li, N., Guo, Y., Wang, X., Yang, S., Huang, M., and Zhang, H. Learning to optimize: Reference vector reinforcement learning adaption to constrained many-objective optimization of industrial copper burdening system. *IEEE Trans. Cybern. 52*, 12 (2022), 12698–12711.

[76] Macrina, G., Di Puglia Pugliese, L., Guerriero, F., and Laporte, G. Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies 120* (2020), 102762.

[77] Malakooti, B., and Raman, V. Clustering and selection of multiple criteria alternatives using unsupervised and supervised neural networks. *J. Intell. Manuf. 11* (2000), 435–453.

[78] Malakooti, B., and Yang, Z. Clustering and group selection of multiple criteria alternatives with application to space-based networks. *IEEE Trans. Syst. Man Cybern. Part B 34*, 1 (2004), 40–51.

[79] Mally, E. Grundgesetze des sollens: Elemente der logik des willens. In *Logische Schriften: Großes Logikfragment, Grundgesetze des Sollens*, K. Wolf and P. Weingartner, Eds. D. Reidel, Dordrecht, 1926, pp. 227–324. Reprint of the original edition published in Graz by Leuschner und Lubensky, Universitäts-Buchhandlung.

[80] McKnight, P. E., and Najab, J. Mann-whitney u test. *The Corsini encyclopedia of psychology* (2010), 1–1.

[81] Meng, C., Todo, Y., Tang, C., Luan, L., and Tang, Z. Dpfsi: A legal judgment prediction method based on deontic logic prompt and fusion of law article statistical information. *Expert Systems with Applications 272* (2025), 126722.

[82] Menéndez, M., Pardo, J., Pardo, L., and Pardo, M. The jensen-shannon divergence. *Journal of the Franklin Institute 334*, 2 (1997), 307–318.

[83] Messaoud, K., Yahiaoui, I., Verroust-Blondet, A., and Nashashibi, F. Attention based vehicle trajectory prediction. *IEEE Transactions on Intelligent Vehicles 6*, 1 (2020), 175–185.

[84] Mirza, M., and Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).

[85] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature 518*, 7540 (2015), 529–533.

[86] Mutzari, D., Deb, T., Molinaro, C., Pugliese, A., Subrahmanian, V.,

and Kraus, S. Defending a city from multi-drone attacks: A sequential stackelberg security games approach. *Artificial Intelligence 349* (2025), 104425.

[87] Neufeld, E. A., Bartocci, E., and Ciabattoni, A. On normative reinforcement learning via safe reinforcement learning. In *International Conference on Principles and Practice of Multi-Agent Systems* (2022), Springer, pp. 72–89.

[88] Neufeld, E. A., Bartocci, E., Ciabattoni, A., and Governatori, G. Enforcing ethical goals over reinforcement-learning policies. *Ethics and Information Technology 24*, 4 (2022), 43.

[89] Olivieri, F., Governatori, G., Cristani, M., Rotolo, A., and Sattar, A. Deontic meta-rules. *Journal of Logic and Computation 34*, 2 (2024), 261–314.

[90] Olszewski, M., Parent, X., and Van der Torre, L. Permissive and regulative norms in deontic logic. *Journal of Logic and Computation* (2023), exad024.

[91] Ozer, M., Keles, I., Toroslu, I. H., and Karagoz, P. Predicting the change of location of mobile phone users. In *Proceedings of the Second ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems* (2013), pp. 43–50.

[92] Ozer, M., Keles, I., Toroslu, İ. H., Karagoz, P., and Ergut, S. Predicting the next location change and time of change for mobile phone users. In *proceedings of the third ACM SIGSPATIAL international workshop on mobile geographic information systems* (2014), pp. 51–59.

[93] Pang, Y., and Liu, Y. Conditional generative adversarial networks (cgan) for aircraft trajectory prediction considering weather effects. In *AIAA Scitech 2020 Forum* (2020), p. 1853.

[94] Papoudakis, G., Christianos, F., Schäfer, L., and Albrecht, S. V. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869* (2021).

[95] Pardalos, P. M., Migdalas, A., and Pitsoulis, L. *Pareto optimality, game theory and equilibria*, vol. 17. Springer Science & Business Media, 2008.

[96] Pledger, T. The role of drones in future terrorist attacks. *Association of the United States Army* (2021).

[97] Poibrenski, A., Klusch, M., Vozniak, I., and Müller, C. M2p3: multimodal multi-pedestrian path prediction by self-driving cars with egocentric vision. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (2020), pp. 190–197.

[98] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision. *CoRR abs/2103.00020* (2021).

[99] Raha, M. H., Deb, T., Rahmun, M., Bijoy, S. A., Firoze, A., and Khan, M. A. Cae: Towards crowd anarchism exploration. In *2020 19th IEEE International*

Conference on Machine Learning and Applications (ICMLA) (2020), IEEE, pp. 559–564.

[100] Rahmun, M., Deb, T., Bijoy, S. A., and Raha, M. H. Uav-crowd: Violent and non-violent crowd activity simulator from the perspective of uav. *arXiv preprint arXiv:2208.06702* (2022).

[101] Raivi, A. M., Huda, S. A., Alam, M. M., and Moh, S. Drone routing for drone-based delivery systems: A review of trajectory planning, charging, and security. *Sensors 23*, 3 (2023), 1463.

[102] Rao, J., Gao, S., Kang, Y., and Huang, Q. Lstm-trajgan: A deep learning approach to trajectory privacy protection. *arXiv preprint arXiv:2006.10521* (2020).

[103] Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research 21*, 1 (2020), 7234–7284.

[104] Ren, Y., Lan, Z., Liu, L., and Yu, H. Emsin: enhanced multi-stream interaction network for vehicle trajectory prediction. *IEEE Transactions on Fuzzy Systems* (2024).

[105] Rönnedal, D. *An introduction to deontic logic*. CreateSpace Independent Publishing Platform, 2010.

[106] Rossiter, A. Drone usage by militant groups: exploring variation in adoption. *Defense & Security Analysis 34*, 2 (2018), 113–126.

[107] Samvelyan, M., Rashid, T., De Witt, C. S., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043* (2019).

[108] ŞEN, O., and AKARSLAN, H. Terrorist use of unmanned aerial vehicles: Turkey's example. *Defence Against Terrorism Review 13* (2020).

[109] Shah, S., Dey, D., Lovett, C., and Kapoor, A. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics* (2018), Springer, pp. 621–635.

[110] Shea-Blymyer, C., and Abbas, H. Generating deontic obligations from utility-maximizing systems. In *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society* (2022), pp. 653–663.

[111] Shea-Blymyer, C., and Abbas, H. Formal ethical obligations in reinforcement learning agents: Verification and policy updates. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (2024), vol. 7, pp. 1368–1378.

[112] Shi, H., Wang, M., and Wang, C. Leader-follower formation learning control of discrete-time nonlinear multiagent systems. *IEEE Trans. Cybern. 53*, 2 (2023), 1184–1194.

[113] Shi, Z., Xu, M., Pan, Q., Yan, B., and Zhang, H. Lstm-based flight trajectory prediction. In *2018 International joint conference on neural networks (IJCNN)* (2018), IEEE, pp. 1–8.

[114] Shukla, P., Shukla, S., and Singh, A. K. Trajectory-prediction techniques for unmanned aerial vehicles (uavs): A comprehensive survey. *IEEE Communications Surveys & Tutorials* (2024).

[115] Sims, A. The rising drone threat from terrorists. *Geo. J. Int'l Aff. 19* (2018), 97.

[116] Stroe, B., Subrahmanian, V., and Dasgupta, S. Optimal status sets of heterogeneous agent programs. In *AAMAS Conference* (2005).

[117] Su, Z., Banerjee, I., and Klabjan, D. Central limit theorems for transition probabilities of controlled markov chains. *arXiv preprint arXiv:2508.01517* (2025).

[118] Subrahmanian, V., Bonatti, P., Dix, J., Eiter, T., Kraus, S., Ross, R., Ozcan, F., and Dix, J. *Heterogeneous agent systems*. MIT press, 2000.

[119] Sutton, R. S., Barto, A. G., et al. *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[120] Taboada, H. A., Baheranwala, F., Coit, D. W., and Wattanapongsakorn, N. Practical solutions for multi-objective optimization: An application to system reliability design problems. *Reliability Engineering & System Safety 92*, 3 (2007), 314–322.

[121] Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. Multiagent cooperation and competition with deep reinforcement learning. *PloS one 12*, 4 (2017), e0172395.

[122] Tedeschi, P., Nuaimi, F. A. A., Awad, A. I., and Natalizio, E. Privacy-aware remote identification for unmanned aerial vehicles: Current solutions, potential threats, and future directions. *IEEE Trans. Ind. Informatics 20*, 2 (2024), 1069–1080.

[123] Tedeschi, P., Sciancalepore, S., and Pietro, R. D. PPCA - privacy-preserving collision avoidance for autonomous unmanned aerial vehicles. *IEEE Trans. Dependable Secur. Comput. 20*, 2 (2023), 1541–1558.

[124] Thompson, M., Tarr, A. A., Tarr, J.-A., and Ritterband, S. Unmanned aerial vehicles: Liability and insurance. In *The Global Insurance Market and Change*. Informa Law from Routledge, 2024, pp. 212–245.

[125] Ullah, F., Sepasgozar, S. M., and Wang, C. A systematic review of smart real estate technology: Drivers of, and barriers to, the use of digital disruptive technologies and online platforms. *Sustainability 10*, 9 (2018), 3142.

[126] Vogel, R. J. Drone warfare and the law of armed conflict. *Denv. J. Int'l L. & Pol'y 39* (2010), 101.

[127] Wan, L., Yuan, J., and Wei, L. Pareto optimization scheduling with two competing agents to minimize the number of tardy jobs and the maximum cost. *Applied Mathematics and Computation 273* (2016), 912–923.

[128] Wang, F.-Y. Agent-based control for networked traffic management systems. *IEEE Intelligent Systems 20*, 5 (2005), 92–96.

[129] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image*

Processing 13, 4 (2004), 600–612.

[130] Wang, Z., Simoncelli, E., and Bovik, A. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003* (2003), vol. 2, pp. 1398–1402 Vol.2.

[131] World Economic Forum. Advanced drone operations: Regulatory challenges and opportunities in cities. WEF Insight Report, 2019. Noting fragmented drone regulations across jurisdictions and lagging legal frameworks.

[132] Wu, C.-W., Shieh, M.-D., Lien, J.-J. J., Yang, J.-F., Chu, W.-T., Huang, T.-H., Hsieh, H.-C., Chiu, H.-T., Tu, K.-C., Chen, Y.-T., Lin, S.-Y., Hu, J.-J., Lin, C.-H., and Jheng, C.-S. Enhancing fan engagement in a 5g stadium with ai-based technologies and live streaming. *IEEE Systems Journal 16*, 4 (2022), 6590–6601.

[133] Yang, W.-C., Marra, G., Rens, G., and De Raedt, L. Safe reinforcement learning via probabilistic logic shields. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence* (2023), pp. 5739–5749.

[134] Yu, L., Zhang, W., Wang, J., and Yu, Y. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence* (2017), vol. 31.

[135] Zeng, W., Chu, X., Xu, Z., Liu, Y., and Quan, Z. Aircraft 4d trajectory prediction in civil aviation: A review. *Aerospace 9*, 2 (2022), 91.

[136] Zhang, Q., and Li, H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput. 11*, 6 (2007), 712–731.

[137] Zhang, Y., Yuan, J., Ng, C. T., and Cheng, T. C. E. Pareto-optimization of three-agent scheduling to minimize the total weighted completion time, weighted number of tardy jobs, and total weighted late work. *Naval Research Logistics 68*, 3 (2021), 378–393.

[138] Zheng, O., Abdel-Aty, M., Yue, L., Abdelraouf, A., Wang, Z., and Mahmoud, N. Citysim: A drone-based vehicle trajectory dataset for safety-oriented research and digital twins. *Transportation Research Record 2678*, 4 (2024), 606–621.

[139] Zhou, S., Yang, L., Liu, X., and Wang, L. Learning short-term spatial–temporal dependency for uav 2-d trajectory forecasting. *IEEE Sensors Journal 24*, 22 (2024), 38256–38269.

[140] Zhu, T., Ye, X., Feng, C., Zhang, Y., Huang, Y., Xu, T., and Chen, E. Diffrntraj: A structure-aware diffusion model for road network-constrained trajectory generation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2024), pp. 4395–4406.

APPENDIX A

A Drone Early Warning System (DEWS) for Predicting Threatening Trajectories

DEWS Features. For each trajectory t, DEWS captures the following types of features. It is important to note that the data used by DEWS was captured in real-time by the Dutch police using third party tools to monitor drone communications. Thus, all of the features listed below are available at or before the time of a flight. Details (including units) are provided in Tables A.1 and A.2.

Basic Features. These include the number of observations, the duration of the flight, the distance traveled, and the communication channel used (e.g. radio-frequency, wifi).

Drone Capability Features. These include the weight, dimensions, max payload, max ascent speed, max descent speed, max horizontal speed, max takeoff altitude, max flight time, max hovering time, max flight distance, max windspeed resistance, max pitch angle, and battery.

Asset Features. Asset features assign values to different parts of a city being protected. They were assigned to regions of the city by Dutch police officers in advance. The features listed here can be extracted using the pre-specified city values and the given drone trajectory. These include the maximum value of assets on the ground that the drone has flown over as well as features depending on the distributions of asset values around each point in the trajectory.

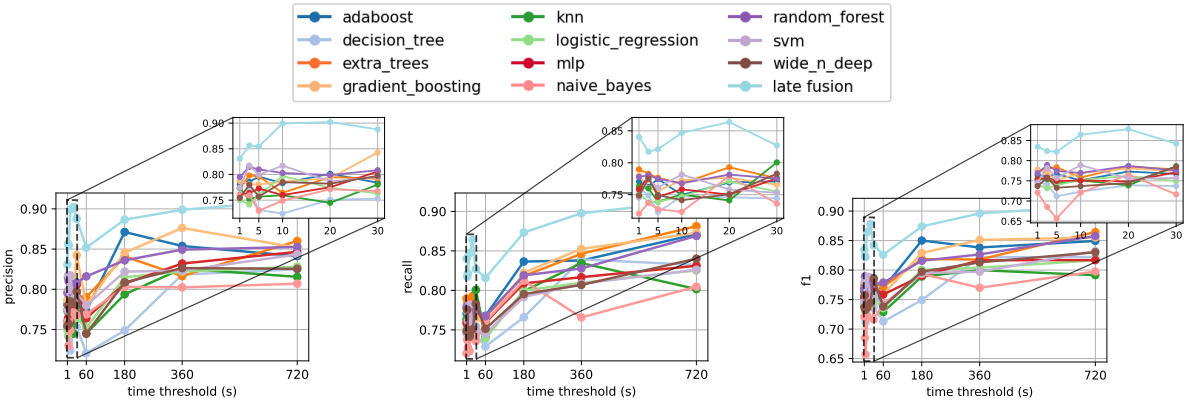


Figure A.1. Low-Threat Prediction (LTP) settings: Precision (a), Recall (b), and F1-score (c) metrics are shown as functions of varying temporal restrictions on the trajectories. The top row provides a zoomed-in view of the results for shorter time windows (less than 30 seconds), while the bottom row displays the complete range of observation windows.

Altitude Features. These include the altitude at both the start and end of the drone's trajectory, the mean altitude throughout the trajectory, the standard deviation of the altitude, and additional metrics based on the distribution of altitude values along the trajectory.

No-fly Zone Features. No-fly Zone features are based on areas within the city where drone flight is restricted. These features include indicators of whether the drone entered a no-fly zone, the percentage of trajectory points within such zones, and features representing the distance from the drone to the nearest no-fly zone.

Speed Features. These include the speed of the drone at both the start and end of its trajectory, the mean speed throughout the trajectory, the standard deviation of the speed, and additional metrics based on the distribution of speed values along the trajectory.

Observation History Features. These include the distance between the current trajectory and the closest (and past) trajectories from the same drone (*self-similarity*) or

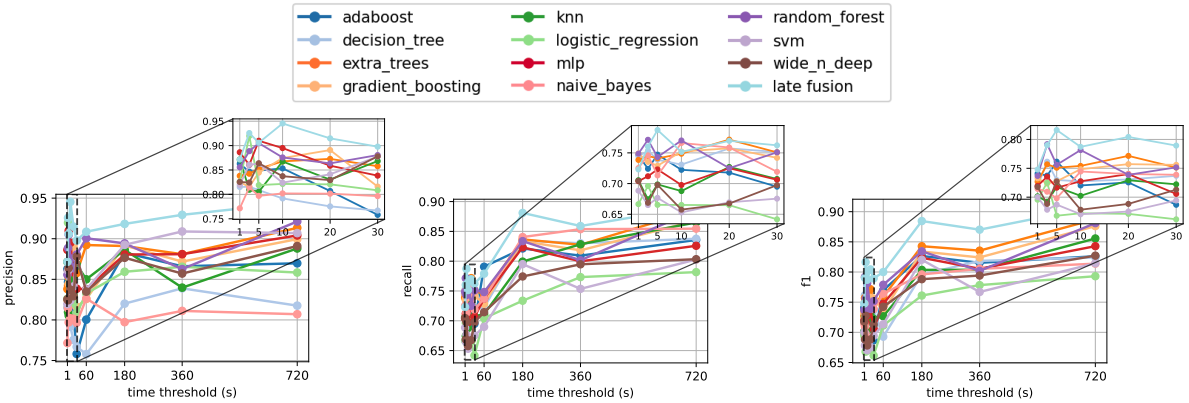


Figure A.2. Medium-Threat Prediction (MTP) settings: Precision (a), Recall (b), and F1-score (c) metrics are shown as functions of varying temporal restrictions on the trajectories. The top row provides a zoomed-in view of the results for shorter time windows (less than 30 seconds), while the bottom row displays the complete range of observation windows.

other observed drones (*cross-similarity*), as well as the threat scores associated with these trajectories.

Early Threat Prediction Evaluation. Figure A.1 illustrates the performance of DEWSunder the Low-Threat Prediction (LTP) setting, while Figure A.2 presents the results for the Medium-Threat Prediction (MTP) setting. For both threat levels, the evaluation metrics — precision, recall, and F1-score — are assessed across different observation windows.

In the LTP scenario, late fusion demonstrates superior performance compared to the 11 classifiers evaluated, achieving high precision, recall, and F1-score across various observation windows. Due to the lower complexity of this setting, high performance is observed early in the trajectories, with the F1-score exceeding 0.85 after just a 10-second observation window.

In the MTP scenario, the performance of DEWS exhibits a similar pattern, with late fusion consistently yielding better results. Precision and recall metrics suggest a

Category	Attribute	Description
Basic	n_records	The total number of recorded data points within the given trajectory.
	duration	The duration (in seconds) from the start to the end of the recorded drone trajectory
	distance	The Haversine distance (in km) traveled by the drone during the recorded trajectory
	communication channel	The type of communication channel used by the drone (e.g., Radio Frequency, Wi-Fi, cellular).
	Weight	The total weight of the drone in grams (g).
	L, w, h	The drone's physical dimensions—length (L), width (w), and height (h) in millimeters (mm).
Capabilities	MaxPayload	The maximum payload capacity the drone can carry, measured in grams (g). It is the weight the drone can safely lift in addition to its own weight.
	MaximumAscentSpeed	The highest speed at which the drone can ascend, measured in meters per second (m/s).
	MaximumDescentSpeed	The highest speed at which the drone can descend, measured in meters per second (m/s).
	MaximumHorizontalSpeed	The maximum speed at which the drone can travel horizontally, measured in meters per second (m/s).
	MaxTakeoffAltitude	The maximum altitude above sea level from which the drone can take off, measured in meters (m).
	MaxFlightTime	The maximum duration the drone can stay airborne on a single battery charge, measured in minutes (min).
	MaxHoveringTime	The maximum time the drone can hover in place, measured in minutes (min).
	MaxFlightDistance	The maximum distance the drone can travel on a single battery charge, measured in kilometers (km).
	MaxWindSpeedResistance	The highest wind speed that the drone can withstand while maintaining stable flight, measured in meters per second (m/s).
	MaxPitchAngle	The maximum angle at which the drone can tilt forward or backward, measured in degrees (°).
	Battery	The battery capacity of the drone, measured in milliampere-hours (mAh).
Assets	av_max	The maximum value of assets on the ground that the drone has flown over.
	$av\_r_h\_b_k$	The proportion of asset values within the h-th radius from each point of the drone's trajectory, distributed across n_{bin} bins. Each $av\_r_h\_b_k$ represents the relative frequency of asset values in the k-th bin. In our experiments, $n_{bin} = 10$, $h \in \{50, 250, 1000\}$ (meters), and $k \in \{1, 2, \ldots, n_{bin}\}$.
	$av\_r_h\_mean$	The mean of asset values within the h-meter radius around each point of the drone's trajectory. $h \in \{50, 250, 1000\}$ (meters).
	$av\_r_h\_std$	The standard deviation of asset values within the h-meter radius around each point of the drone's trajectory. $h \in \{50, 250, 1000\}$ (meters).

Table A.1. DEWS Features categories and descriptions (Part 1).

well-calibrated system, with precision reaching approximately 0.95 after a 10-second observation window. Nonetheless, recall is considerably lower compared to the LTP setting, reflecting the increased challenge of identifying threatening trajectories in this more complex scenario.

Category	Attribute	Description
Altitude	h_start	The altitude at the beginning of the drone's trajectory, measured as the height above the take-off point.
	h_end	The altitude at the end of the drone's trajectory, measured as the height above the take-off point.
	h_mean	The mean altitude throughout the drone's trajectory, measured as the average height above the take-off point.
	h_std	The standard deviation of the altitude throughout the drone's trajectory.
	h_b_k	The proportion of altitude values within each of the n_{bin} bins along the drone's trajectory. Each h_b_k represents the relative frequency of altitude values in the k-th bin. In our experiments, $n_{bin} = 10$ and $k \in \{1, 2, \ldots, n_{bin}\}$.
No-fly Zones	enter_noflyzone	A boolean value indicating whether the drone entered any no-fly zones during its trajectory.
	perc_noflyzone	The percentage of records (points in the trajectory) where the drone was in a no-fly zone.
	nf_d_min	The minimum distance from the drone to the nearest no-fly zone during its trajectory.
	nf_d_max	The maximum distance from the drone to the nearest no-fly zone during its trajectory.
	nf_d_mean	The average distance from the drone to the nearest no-fly zone during its trajectory.
	nf_d_std	The standard deviation of distances from the drone to the nearest no-fly zone during its trajectory.
Speed	sp_start	The speed of the drone at the start of the trajectory, measured in kilometers per hour (km/h).
	sp_end	The speed of the drone at the end of the trajectory, measured in kilometers per hour (km/h).
	sp_mean	The average speed of the drone during the trajectory, calculated as the mean of the speeds between consecutive points.
	sp_std	The standard deviation of the drone's speed over the recorded trajectory.
	sp_b_k	The proportion of speed values that fall into each of the n_{bin} bins. Each sp_b_k represents the relative frequency of speeds in the k-th bin. In our experiments, $n_{bin} = 10$ and $k \in \{1, 2, \ldots, n_{bin}\}$.
Observation History	self_sim_k_i	The distance between the current trajectory and the i-th closest (and past) trajectory from the same drone, based on a specified distance metric. In our experiments, we use cosine similarity and $i \in \{1, 2, \ldots, k\}$.
	self_threat_k_i	The threat score associated with the i-th closest trajectory from the same drone.
	cross_sim_k_i	The distance between the current trajectory and the i-th closest (and past) trajectory from different drones, based on a specified distance metric. In our experiments, we use cosine similarity and $i \in \{1, 2, \ldots, k\}$.
	cross_threat_k_i	The threat score associated with the i-th closest trajectory from other drones.

Table A.2. DEWS Features categories and descriptions (Part 2).

APPENDIX B

Declarative Logic-based Pareto-Optimal Agent Decision Making

B.1. Proofs

Proof of pro:closure-complexity. In the following, the worst-case time complexity is always understood. Lines 1–4 can be executed as follows: the status atoms in SS are sorted by their action α, and then the resulting sorted list is scanned checking the condition of the **for each** loops for each traversed element—checking such a condition for a single element can now be done in constant time, since there is a constant number of status atoms with the same action. Assuming that the addition of a new element to SS takes constant time (e.g., using a list), the overall time taken by lines 1–4 is $O(|SS| \cdot lg|SS|)$. The same reasoning applies to lines 5–6, even though the *updated* set SS needs to be traversed, whose cardinality is at most three times the cardinality of the original set SS, and thus the overall time taken by lines 5–6 is $O(|SS| \cdot lg|SS|)$ too.

On line 7, condition (i) can be checked in $O(|SS| \cdot lg|SS|)$ time (again, by first sorting SS as discussed above), while condition (ii) can be checked in $O(|SS| \cdot |S_t|)$ time (here we are considering the size of $Pre(\alpha)$ to be a constant, as the set of actions is fixed). Line 8 takes constant time.

Line 9 takes $O(|SS| \cdot ||DC||)$ time. Lines 10–11 take constant time.

We now consider lines 12–26, which consist of two nested loops. The number of times lines 15–25 are executed is $O(|A| \cdot g_P)$, because the outer loop can make at most $|A|$ iterations, and the inner loop clearly makes g_P iterations. Let us focus on the complexity of lines 15–25 (when executed once). Line 15 takes constant time. Line 16 takes $O(\chi_P \cdot |S_t| + b_P \cdot |SS'|)$ time. Lines 17–21 take constant time (again, here we consider an addition to SS' to take constant time). On line 22, condition (i) can be checked in $O(|SS'| \cdot lg|SS'|)$ time, while condition (ii) can be checked in $O(|SS'| \cdot |S_t|)$ time. Line 23 takes constant time. Line 24 takes $O(|SS'| \cdot ||DC||)$ time. Line 25 takes constant time. So, the overall time complexity of lines 12–26 is $O(|A| \cdot g_P \cdot (\chi_P \cdot |S_t| + b_P \cdot |SS'| + |SS'| \cdot lg|SS'| + |SS'| \cdot |S_t| + |SS'| \cdot ||DC||))$, which can be rewritten as $O(|A| \cdot g_P \cdot (\chi_P \cdot |S_t| + |SS'| \cdot (b_P + lg|SS'| + |S_t| + ||DC||)))$. Notice that $|SS'|$ is $O(|A|)$. Thus, the overall time complexity of lines 12–26 can be rewritten as $O(|A| \cdot g_P \cdot (\chi_P \cdot |S_t| + |A| \cdot (b_P + lg|A| + |S_t| + ||DC||)))$.

Line 27 takes constant time.

From the analysis above, the worst-case time complexity of poss:alg:closure is $O(|A| \cdot g_P \cdot (\chi_P \cdot |S_t| + |A| \cdot (b_P + lg|A| + |S_t| + ||DC||)))$. □

Proof of th:baseline-complexity. In the following, the worst-case time complexity is always understood. The worst-case time complexity of line 2 is as per pro:closure-complexity. Lines 3–4 take constant time. Lines 5–7 take $O(|A| \cdot (|S_t| + g_P))$ time, since the cardinality of LSS is $O(g_P)$. Notice that $|SA|$ is $O(|A|)$. Line 8 takes constant time. Checking whether a status set SS is feasible as per def:feasibleSS takes $O(|SS| \cdot lg|SS| + |SS| \cdot |S_t| + g_P \cdot (|S_t| \cdot \chi_P + |SS| \cdot b_P) + |SS| \cdot ||AC|| + |S_t| \cdot ||IC|| + f_{\mathsf{conc}}(|SS|, |S_t|))$ time. Lines 9–11 take $O(2^{|A|} \cdot (|A| \cdot lg|A| + |A| \cdot |S_t| + g_P \cdot (|S_t| \cdot \chi_P + |A| \cdot b_P) + |A| \cdot ||AC|| +$

$|S_t| \cdot ||IC|| + f_{\mathsf{conc}}(|A|, |S_t|)))$ time, since, for any status set SS s.t. $LSS \subseteq SS \subseteq SA$, we have $|SS| = O(|A|)$. Lines 12–13 take constant time. Lines 14–15 take $O(2^{2^{|A|}} \cdot f_{OF}(|A|))$. From the analysis above, the overall (worst-case) time complexity of poss:alg:poss-naive is $O(|A|^2 \cdot g_P \cdot ||DC|| + 2^{2^{|A|}} \cdot f_{OF}(A) + 2^{|A|} \cdot (|A| \cdot lg|A| + |A| \cdot |S_t| + g_P \cdot (|S_t| \cdot \chi_P + |A| \cdot b_P) + |A| \cdot ||AC|| + |S_t| \cdot ||IC|| + f_{\mathsf{conc}}(|A|, |S_t|)))$, where $|A|^2 \cdot g_P \cdot ||DC||$ is the part of the complexity of poss:alg:closure (see line 2) that is not dominated by the complexity of the rest of poss:alg:poss-naive. □

APPENDIX C

GUARDIAN: Governance-Unified Aerial Reinforcement-Defense In Accordance with Norms

C.1. Structure of GUARDIAN

We now provide the formal underpinnings of GUARDIAN.

C.1.1. Drones

Each drone operates within the $M \times N$ city grid G.

C.1.1.1. Static Properties. Each drone d possesses inherent static properties that define its capabilities:

- **Drone ID** (id_d): A unique identifier assigned from $\mathbb{N}$.
- **Team ID** (team_d): Indicates team affiliation. We denote BLUE team as $\text{team}_d = 1$ and RED team as $\text{team}_d = 2$.
- **View Range** (r_d): A positive integer $r_d \in \mathbb{N}$ defining the radius within which the drone can observe its surroundings.
- **Fire Range** (f_d): A positive integer $f_d \in \mathbb{N}$ specifying the maximum distance at which the drone can hit a target.
- **Cost** (cost_d): cost of the drone.

C.1.1.2. Dynamic Properties. Each drone d has different dynamic properties, that is, the property values change over time.

- **Position** ($\mathbf{x}_d(t)$): Current location on the grid at time t with $\mathbf{x}_d(t) \in G$. In other words, we unify the row and column notation, so $\mathbf{x}_d(t)$ indicates the 2D cell location.
- **Battery** ($B_d(t)$): Remaining battery at time t, with $B_d(t) \geq 0$. The drone is considered non-operational if $B_d(t) = 0$. This property is dynamic and may decrease over time (we assume no recharging).
- **Payload** ($p_d(t)$): Remaining ammunition at time t, with $p_d(t) \geq 0$. We assume payload is also non-increasing over time (no reloads).
- **Reward** ($R_d(t)$): Accumulated reward up to time t, with $R_d(t) \in \mathbb{R}$.

C.1.2. State Space of Drones

Actual State $S_d(t)$ contains all internal and external attributes of drone d at time t, including confidential information not observable by other agents (drones or CCTV cameras). The drone sends this information to the HQ directly. We define:

$$S_d(t) = \begin{bmatrix} \mathrm{id}_d \\ \mathrm{team}_d \\ r_d \\ f_d \\ \mathrm{cost}_d \\ \mathbf{x}_d(t) \\ B_d(t) \\ p_d(t) \\ R_d(t) \end{bmatrix}.$$

Public State $S_d^{\mathrm{public}}(t)$ includes only the attributes that are externally observable by other agents, excluding any confidential or internal information. The public state of drone d at time t is:

$$S_d^{\text{public}}(t) = \begin{bmatrix} \text{id}_d \\ \text{team}_d \\ \text{cost}_d \\ B_d(t) \\ p_d(t) \\ \mathbf{x}_d(t) \end{bmatrix}.$$

C.1.3. Observation Space of Drones

Drone d observes other drones and cameras within its view range r_d. The set of neighboring entities (drones or CCTV cameras) at time t is:

$$N_d(t) = \{b \mid b \neq d,\ \|\mathbf{x}_d(t) - \mathbf{x}_b(t)\| \leq r_d\},$$

where $\|\cdot\|$ is a distance metric, which stays consistent over time.

The drone's observation at time t is:

$$O_d(t) = \left\{ S_b^{\text{public}}(t) \mid b \in N_d(t) \right\} \tag{C.1}$$

where $S_b^{\text{public}}(t)$ includes publicly observable information about agent b. In other words, the drone can observe all publicly available state of drones or CCTVs within its view range—the public state of CCTV cameras will be defined in the following.

C.1.4. Action Space

The set of possible actions for drone d at time t, denoted $\mathcal{A}_d(t)$, includes:

- **Movement Actions**:
  - Move Up: $MoveTo_d$(up)
  - Move Down: $MoveTo_d$(down)
  - Move Left: $MoveTo_d$(left)
  - Move Right: $MoveTo_d$(right)

  The drone moves one cell in the specified direction.
- **Fire at drone**: $FireAtDrone_d(d')$, where d' is a drone within fire range f_d.
- **Fire at CCTV**: $FireAtCCTV_d(c)$, where c is a CCTV within fire range f_d.
- **Fire at Cell**: $FireAtCell_d$, which attempts to destroy the cell (setting its value to zero) currently occupied by the drone.
- **No Operation**: $NoOp_d$

The outcome of firing actions, whether at drones or CCTV cameras or cells, is determined probabilistically, drawn from predefined distributions. Various distributions for modeling these probabilistic outcomes are supported by our testbed.

C.1.5. Preconditions of Actions

To execute an action, the corresponding preconditions must be met:

Movement Preconditions. Moving to cell i, j requires:

- $(i, j) \in G$ (within grid bounds).
- Target cell is vacant: no agent occupies (i, j).

Fire at Target Preconditions. Firing at target b (drone or CCTV camera) requires:

- Target within fire range: $\|\mathbf{x}_d(t) - \mathbf{x}_b(t)\| \leq f_d$.
- Sufficient payload: $p_d(t) > 0$.

Fire at Cell Preconditions.

- Drone d is in cell (i, j) to be destroyed: $\mathbf{x}_d(t) = (i, j)$.
- Sufficient payload: $p_d(t) > 0$.

C.1.6. State Transitions

The following are the drone state updates based on actions:

- **Movement**: $\mathbf{x}_d(t+1) = (i, j)$ where (i, j) is the new location of the drone after performing a move action.
- **Firing at Agent** b: $\begin{aligned} p_d(t+1) &= p_d(t) - 1, \\ B_b(t+1) &= B_b(t) - 1 \end{aligned}$ where the target drone's battery decreases by 1, indicating damage.
- **Firing at Cell**: $\begin{aligned} p_d(t+1) &= p_d(t) - 1, \\ v_{i,j}(t+1) &= 0 \end{aligned}$ where $(i, j) = \mathbf{x}_d(t)$ (cell destroyed).

Action Outcome Distribution. The testbed supports stochastic action outcomes through three configurable modes:

(1) **Deterministic**: All actions succeed with probability $p = 1$.

(2) **Uniform stochastic**: Success probability $p = 0.5$, determined by sampling $U \sim \text{Uniform}(0, 1)$ and succeeding if $U > 0.5$.

(3) **Normal stochastic**: Success probability $p = 0.5$, determined by sampling $Z \sim \mathcal{N}(0, 1)$ and succeeding if $Z > 0$.

All experiments reported in this chapter use the deterministic mode ($p = 1$), ensuring the state transitions defined above occur with certainty.

C.1.7. CCTV Cameras

CCTV cameras are stationary agents deployed by the BLUE Team to monitor the city grid G.

C.1.7.1. Static Properties. Each CCTV camera c has:

- **CCTV ID** (id_c): Unique identifier from $\mathbb{N}$.
- **Team ID** (team_c): All CCTV cameras belong to the BLUE team, i.e., $\text{team}_c = 1$.
- **Position** ($\mathbf{x}_c$): Fixed at $\mathbf{x}_c \in G$.
- **View Range** (r_c): A positive integer $r_c \in \mathbb{N}$.
- **Cost** (cost_c): cost of the camera.

Dynamic Properties. At time t, a CCTV camera c can be either

- *alive*, which holds if $v_{\mathbf{x}_c}(t) > 0$ (in which case we set $\text{alive}_c(t)$ to true), or
- *destroyed*, which holds otherwise (in which case $\text{alive}_c(t)$ is false).

C.1.8. State Space of CCTVs

Actual State of a CCTV camera c at time t is defined as:

$$S_c(t) = \begin{bmatrix} \text{id}_c \\ \text{team}_c \\ \mathbf{x}_c \\ r_c \\ \text{cost}_c \\ \text{alive}_c(t) \end{bmatrix}.$$

The public state of a CCTV camera c at time t is:

$$S_c^{\text{public}}(t) = \begin{bmatrix} \text{id}_c \\ \text{team}_c \\ \mathbf{x}_c \\ \text{cost}_c \\ \text{alive}_c(t) \end{bmatrix}.$$

Here, the view range r_c is the only non-shared confidential information, hence is not a part of the public state.

C.1.9. Observation Space of CCTVs

Like drones, the observations of a CCTV camera c are based on view range r_c, so we define the set of neighboring agents of c at time t as:

$$N_c(t) = \{b \mid \|\mathbf{x}_c - \mathbf{x}_b(t)\| \leq r_c\},$$

and c's observation at time t as:

$$O_c(t) = \left\{S_b^{\text{public}}(t) \mid b \in N_c(t)\right\}.$$

C.1.10. Autonomous Headquarters (HQ)

In GUARDIAN, the HQ of each team is responsible for overseeing the deployment and operations of its drones and, for the BLUE Team, CCTV cameras. By leveraging these resources, HQs play a critical role in decision-making The HQ interacts with its agents and the environment to achieve the team's objectives: maximizing damage (RED Team) or minimizing damage (BLUE Team). In summary, the HQ's responsibilities include:

- **Strategic Decision-Making**: Formulating strategies to achieve team objectives, such as defending key areas or maximizing damage.
- **Resource Management**: Allocating resources like drones and payloads efficiently.
- **Drone Coordination**: Assigning actions to drones, monitoring drone status, and integrating observations.

C.1.10.1. Static Properties. Each team k has a Headquarters (HQ), where $k = 1$ denotes BLUE and $k = 2$ denotes RED. The HQ is defined by the following static properties:

- **Team ID** (team_k): Team identifier, where $\text{team}_k = k$.
- **Number of Drones** (D_k): The initial number of drones deployed by the team, where $D_k \geq 1$
- **Number of CCTV Cameras** (C_k): The number of CCTV cameras deployed by team k. Only BLUE deploys cameras, so $C_1 \geq 0$ and $C_2 = 0$.

C.1.10.2. Dynamic Properties. At each time t, the HQ's dynamic properties evolve based on interactions with agents and the environment:

- **Team Reward** ($R_k(t)$): The cumulative reward accrued by the team up to time t, $R_k(t) \in \mathbb{R}$
- **Set of Drones** ($\mathcal{D}_k(t)$): The collection of drones controlled by the HQ that are operational at time t.
- **Set of CCTV Cameras** ($\mathcal{C}_k(t)$): The collection of operational CCTV cameras at time t. For RED, $\mathcal{C}_2(t) = \emptyset$.

C.1.10.3. Observation Space. Each HQ aggregates observations from its drones and CCTV cameras to form a comprehensive view of the environment:

$$O_k^{\text{HQ}}(t) = \bigcup_{n \in \mathcal{D}_k(t) \cup \mathcal{C}_k(t)} O_n(t)$$

where:

- $\mathcal{D}_k(t) \cup \mathcal{C}_k(t)$: The set of all drones and CCTV cameras belonging to team k at time t.
- $O_n(t)$: The observation received from entity n at time t, where n is either a drone or a CCTV camera.

C.1.10.4. State Space. The state of the Headquarters at time t is represented by:

$$S_k^{\text{HQ}}(t) = \begin{bmatrix} O_k^{\text{HQ}}(t) \\ \{S_d(t)\}_{d \in \mathcal{D}_k(t)} \\ \{S_c(t)\}_{c \in \mathcal{C}_k(t)} \end{bmatrix} \tag{C.2}$$

where $S_d(t)$ and $S_c(t)$ are the state of drone and CCTV camera at time t, as previously defined.

C.1.11. Action Space

The HQ's action space only involves assigning actions to drones. For each drone $d \in \mathcal{D}_k(t)$, at time t the HQ selects an action $a_d(t)$ from the drone's action space $\mathcal{A}_d(t)$.

C.1.12. Modeling Communication Consistency

Communication between drones and their HQ occurs at each time step but may experience random failures. We model communication success using a Bernoulli random variable:

$$C_d(t) \sim \text{Bernoulli}(p_{\text{comm}}) \tag{C.3}$$

where $C_d(t) = 1$ indicates successful communication and $C_d(t) = 0$ indicates failure. The parameter p_{comm} represents communication reliability. When $C_d(t) = 1$, communication is consistent, and the drone receives the action from the HQ. When $C_d(t) = 0$, communication fails, and the drone does not receive any action from the HQ or send information to the HQ.

C.2. Incorporating Ethical/Legal Norms in GUARDIAN Framework

C.2.1. State Representation: Atoms and Predicates

We define a set of predicate symbols to represent the environment's state and drone d's possible actions. **Note:** The essential predicates used in the deontic rules are defined in Section 5.3.2. This section provides the complete list of all predicates used in GUARDIAN for reference.

C.2.1.1. Predicates for Drones.

- *blue*(d): Drone d is on the BLUE (defending) Team.
- *red*(d): Drone d is on the RED (attacking) Team.
- *position*(d, i, j, t): Drone d is located in cell (i, j) at time t.
- *InFireRange*(d, d'): Drone d' is within the firing range of drone d.
- *HasPayload*(d): Drone d still has nonzero ammunition.
- *ImmediateThreat*(d'): If the observed state of drone d' on the opposing team contains $B_{d'} > 0$ and $p_{d'} > 0$, then drone d' is deemed an immediate threat. This is determined from publicly observable state information.
- *SameTeam*(d, d'): Drone d' is on the same team as drone d.
- *killed*(d, d', t): True if drone d' is eliminated by drone d at time t.
- *surv*(d, t): True if drone d survives time step t.
- *resp*(d, c, t): True if drone d is responsible for protecting cell c at time t.
- *fired*(d, t): True if drone d fires at time t.

C.2.1.2. Predicates for HQ.

- *AssignedAction*(d, a, t): At time t, the BLUE HQ has assigned action a to drone d.
- *CommConsistent*(d, t): Communication between drone d and the HQ is functioning properly at time t (no packet loss or jamming).

C.2.1.3. Predicates for Environments.

- *Adjacent*(i, j, p, q): Cell (p, q) is orthogonally adjacent (up, down, left, right) to cell (i, j), i.e., the four cardinal neighbors of (i, j) are $(i–1, j)$, $(i+1, j)$, $(i, j–1)$, and

$(i, j{+}1)$ whenever they lie within the grid bounds. Formally, $(|i-p|+|j-q|=1)$ and $1 \leq p \leq M$, $1 \leq q \leq N$ [1].

- *Utility*(i, j, u, t): Cell (i, j) has utility u at time t, i.e. $u = v_{i,j}(t)$.
- *CivilianArea*(i, j): Cell (i, j) is known to be a protected civilian region. BLUE drones must be extremely cautious about firing here.
- *alive*(c, t): True if cell c (or the grid cell containing CCTV c) has not been destroyed by time t, i.e., $v_c(t) > 0$.

C.2.1.4. Predicates for Derived Utility. Certain norms below compare the utility of one cell against another. We introduce two parameters λ and λ', where $1 < \lambda < \lambda'$. They determine how we compare the neighboring cells' utilities to $v_{i,j}(t)$.

Intuitively:

- λ is a moderate threshold (e.g., 1.2 or 1.3). If all neighbors are $\geq \lambda \times v_{i,j}(t)$, or if a neighbor is $\geq \lambda' \times v_{i,j}(t)$, then the risk of letting the RED drone move is high.
- λ' is a higher threshold (e.g., 2.0). Even if some neighbors are only moderately above $v_{i,j}(t)$, we become very concerned if at least one neighbor exceeds $\lambda' \times v_{i,j}(t)$.

We define:

- *HasLowerUtilityNeighbor*(i, j, t, λ): Cell (i, j) at time t has at least one neighbor with strictly lower utility, and no neighbor exceeds $\lambda \times v_{i,j}(t)$:

$$\exists (p, q) : Adjacent(i, j, p, q) \wedge v_{p,q}(t) < v_{i,j}(t),$$

$$\forall (p', q') : Adjacent(i, j, p', q') \Rightarrow v_{p',q'}(t) < \lambda\, v_{i,j}(t).$$

[1] here we assume 4-neighbor adjacency, leaving diagonal adjacency extension as future work

- *AllNeighborsAbove*(i, j, t, λ): Every neighbor (p, q) of (i, j) has utility $\geq \lambda\, v_{i,j}(t)$:

$$\forall (p, q) : \mathit{Adjacent}(i, j, p, q) \Rightarrow v_{p,q}(t) \geq \lambda\, v_{i,j}(t).$$

- *HighValueNeighbor*(i, j, t, λ'): At least one neighbor (p, q) of (i, j) has utility $\geq \lambda'\, v_{i,j}(t)$:

$$\exists (p, q) : \mathit{Adjacent}(i, j, p, q) \wedge v_{p,q}(t) \geq \lambda'\, v_{i,j}(t).$$

The parameters $\lambda > 1$ and $\lambda' > \lambda$ are chosen by domain experts (e.g., $\lambda = 1.1$, $\lambda' = 2.0$) to decide whether adjacent cells are sufficiently high-value compared to (i, j).

C.2.1.5. Predicates for Actions. For drone d, we unify all action symbols by using d as subscript:

- $\mathit{FireAtDrone}_d(d')$: Drone d fires at drone d'.
- $\mathit{FireAtCCTV}_d(c)$: Drone d fires at CCTV c.
- $\mathit{FireAtCell}_d(i, j)$: Drone d fires at cell (i, j).
- $\mathit{MoveTo}_d(i, j)$: Drone d moves to cell (i, j).
- *AssignedAction*(d, a): The HQ has suggested action a to drone d.
- *CommConsistent*(d): Communication between drone d and its HQ is functioning.
- $\mathit{ExecuteAssignedAction}_d(a)$: Suggested action a is executed.

Throughout the subsequent formulations, each ground action α_d will be taken from drone d's *action space* at time t, denoted by $\mathcal{A}_d(t)$.

C.2.2. Formulating Ethical Norms

Using the deontic operators and the predicates defined, we formalize ethical norms $\mathcal{N}_d$ as operating rules of drone d for the ethical compliance verification process. The formal specification of all eight deontic rules is presented in Section 5.3.2. Below we reproduce the formal specification for reference and provide additional context for each norm.

Formal Specification of Norms $\mathcal{N}_d$ as Deontic Rules

Norm 1: **Never firing at cell:** $\mathbf{F}\mathit{FireAtCell}_d(i,j) \leftarrow \mathit{blue}(d)$

Norm 2: **Prohibition of firing at civilian areas:** $\mathbf{F}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{CivilianArea}(i,j) \wedge \mathit{InFireRange}(d,d') \wedge \neg \mathit{ImmediateThreat}(d')$

Norm 3: **Obligation to follow HQ orders (if communication is consistent):** $\mathbf{O}\mathit{ExecuteAssignedAction}_d(a) \leftarrow \mathit{blue}(d) \wedge \mathit{CommConsistent}(d) \wedge \mathit{AssignedAction}(d,a)$

Norm 4: **Prohibition of friendly fire:** $\mathbf{F}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{blue}(d') \wedge \mathit{SameTeam}(d,d')$

Norm 5: **Permission to engage a red drone in a civilian area (under threat):** $\mathbf{P}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{InFireRange}(d,d') \wedge \mathit{CivilianArea}(i,j) \wedge \mathit{ImmediateThreat}(d')$

Norm 6: **Forbid firing if RED drone is not an immediate threat and a lower-utility neighbor exists:** $\mathbf{F}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{InFireRange}(d,d') \wedge \neg \mathit{ImmediateThreat}(d') \wedge \mathit{HasLowerUtilityNeighbor}(i,j,t,\lambda)$

Norm 7: **Obligated to engage a threat if all neighbors are higher utility:** $\mathbf{O}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{InFireRange}(d,d') \wedge \mathit{ImmediateThreat}(d') \wedge \mathit{AllNeighborsAbove}(i,j,t,\lambda)$

Norm 8: **Obligated to engage a threat if any neighbor is extremely high-value:** $\mathbf{O}\mathit{FireAtDrone}_d(d') \leftarrow \mathit{blue}(d) \wedge \mathit{red}(d') \wedge \mathit{position}(d,i,j,t) \wedge \mathit{position}(d',i,j,t) \wedge \mathit{InFireRange}(d,d') \wedge \mathit{ImmediateThreat}(d') \wedge \mathit{HighValueNeighbor}(i,j,t,\lambda')$

Norm 1: Never firing at cell. A BLUE drone is always forbidden from deliberately firing on a cell (i, j).

Norm 2: Prohibition of firing at civilian areas. The BLUE drone must refrain from firing over a cell (i, j) designated as a civilian area to avoid damage. For example, if d and d' are co-located in a hospital cell, the BLUE drone is forbidden from engaging.

Norm 3: Obligation to follow HQ orders (if communication is consistent). The BLUE drone must comply with HQ instructions if the communication channel is reliable at that step, ensuring centralized coordination.

Norm 4: Prohibition of friendly fire. A BLUE drone must never fire at another BLUE drone.

Norm 5: Permission to engage a RED drone in a civilian area (under threat). If a RED drone is an immediate threat inside a civilian area, the BLUE drone may fire to prevent severe harm.

Norm 6: Forbid firing if RED drone is not an immediate threat and a lower-utility neighbor exists. If the RED drone is not evidently threatening, and there is a less valuable neighboring cell, the BLUE drone should not fire, in hopes the RED drone moves to that location.

Norm 7: Obligated to engage a threat if all neighbors are higher utility. If all neighboring cells are more valuable and the RED drone is an immediate threat, the BLUE drone must fire to prevent the threat from moving to those cells.

Norm 8: Obligated to engage a threat if any neighbor is extremely high-value. If any neighboring cell is extremely critical and the RED drone is an immediate threat, the BLUE drone must fire to prevent catastrophic damage.

C.2.3. Integrity Constraints

Integrity constraints (ICs) are conditions that must always hold to maintain system consistency and safety. Their satisfaction is guaranteed by the computation of *feasible* status sets. Below are illustrative ICs, whose intuitive meaning is that the conjunction on the right-hand side of $\leftarrow$ must be false in order for the IC to be satisfied.

IC_1: Engagement Within Firing Range. Drone d cannot fire at another drone d' unless the target is within its firing range:

$$\leftarrow \ \mathit{FireAtDrone}_d(d') \ \wedge \ \neg \ \mathit{InFireRange}(d, d').$$

IC_2: Adequate Payload Requirement. Drone d cannot fire if it lacks sufficient payload:

$$\leftarrow \ \mathit{FireAtDrone}_d(d') \ \wedge \ \neg \ \mathit{HasPayload}(d).$$

C.2.4. Action Constraints

Action constraints (ACs) define permissible combinations of concurrent actions within a single time step for drone d. Below is an illustrative constraint.

AC_1: Single Target Engagement. Drone d cannot engage multiple targets simultaneously:

$$\leftarrow \ \mathit{FireAtDrone}_d(d_1) \ \wedge \ \mathit{FireAtDrone}_d(d_2) \ \wedge \ d_1 \neq d_2.$$

The logic here might reflect a more detailed temporal separation, but in a single-step concurrency model, we encode it as a denial constraint of performing both at once.

C.2.5. Status Sets and Feasibility

To ensure that drone d acts ethically, we verify whether its intended actions are part of a *feasible status set*. A status set SS_d is a set of ground status atoms representing the deontic statuses of drone d's actions. A status set is *feasible* if it satisfies the following conditions (44):

(1) $\mathbf{O}\alpha_d \in SS_d \implies \mathbf{P}\alpha_d \in SS_d$.

(2) $\mathbf{O}\alpha_d \in SS_d \implies \mathbf{Do}\alpha_d \in SS_d$.

(3) $\mathbf{Do}\alpha_d \in SS_d \implies \mathbf{P}\alpha_d \in SS_d$.

(4) $\mathbf{P}\alpha_d \in SS_d \implies \mathbf{F}\alpha_d \notin SS_d$.

(5) $\mathbf{P}\alpha_d \in SS_d \implies$ the preconditions of α_d are satisfied in $S_d(t)$.

(6) SS_d is closed under drone d's operating rules (i.e., if a rule's body is satisfied, its head is in SS_d).

(7) The set of actions $\mathcal{A}^{\mathbf{Do}}(SS_d) = \{\alpha_d \mid \mathbf{Do}\alpha_d \in SS_d\}$ satisfies the action constraints AC.

(8) The resulting state after executing $\{\alpha_d \mid \mathbf{Do}\alpha_d \in SS_d\}$ satisfies the integrity constraints IC.

The set $\mathcal{A}^{\mathbf{Do}}(SS_d)$ represents the *executable set of concurrent actions* that is ethically consistent with the norms.

C.2.6. Ethically Feasible Status Set Computation Algorithm

We now present the procedure for computing drone d's *feasible status set* at a given time step. More specifically, we leverage (36) for feasible status set computation. Our approach relies on two algorithms:

(1) A *least status set* generation (Algorithm 11), which initializes and expands a status set until it attains a stable status, and

(2) An *ethical status set* search (Algorithm 12), which systematically enumerates and checks candidate status sets for feasibility.

Given a set A of actions, we define $SA(A) = \{\, Op\, \alpha \mid \alpha \in A \text{ and } Op \in \{\mathbf{F}, \mathbf{P}, \mathbf{O}, \mathbf{Do}\}\}$ to be the set of all possible status atoms over actions in A.

Here, Algorithm 11 (LSS) starts with a set of status atoms SS and incrementally applies drone d's operating rules. Whenever $\mathbf{O}\alpha_d$ is inferred, it triggers the inclusion of $\mathbf{P}\alpha_d$ and $\mathbf{Do}\alpha_d$. Similarly, if $\mathbf{Do}\alpha_d$ is inferred, then $\mathbf{P}\alpha_d$ must also be included. During the closure process, if any contradiction arises (e.g., both $\mathbf{P}\alpha_d$ and $\mathbf{F}\alpha_d$ appear, or an action α_d is permitted even though its preconditions are not satisfied), the algorithm returns $\perp$, indicating that no feasible set can be formed.

Then, Algorithm 12 uses the result of the LSS process as a baseline and explores different ways of assigning the "do" status $\mathbf{Do}\alpha_d$ for those actions α_d not precluded by $\mathbf{F}\alpha_d$ or unsatisfied preconditions. The algorithm systematically expands candidate status sets, checks action constraints (e.g., no simultaneous movement and firing), and enumerates only those sets that remain free of contradictions. The procedure returns a collection of at most τ *ethically feasible* status sets (possibly none), each of which specifies a valid concurrency of drone d's actions in the current time step.

If Algorithm 11 (LSS) returns $\perp$, it means there is *no* ethically compliant action space for drone d in the current state including SS; in that scenario, d may perform *no operation* or revert to a default fallback behavior. If Algorithm 12 succeeds, it outputs a set of feasible status sets, each of which represents one valid concurrency option d can

Algorithm 11 *LSS*: Least Status Set Algorithm for Drone d

Input: A status set SS, drone state $S_d(t)$, norms $\mathcal{N}_d$, and a set DC of denial action constraints for drone d.

Output: A status set SS_d or $\perp$.

for each $\mathbf{O}\alpha_d \in SS$ s.t. $\mathbf{P}\alpha_d \notin SS$ **do**
    Add $\mathbf{P}\alpha_d$ to SS.
end for
for each $\mathbf{O}\alpha_d \in SS$ s.t. $\mathbf{Do}\alpha_d \notin SS$ **do**
    Add $\mathbf{Do}\alpha_d$ to SS.
end for
for each $\mathbf{Do}\alpha_d \in SS$ s.t. $\mathbf{P}\alpha_d \notin SS$ **do**
    Add $\mathbf{P}\alpha_d$ to SS.
end for
if there exists α_d s.t. (i) $\{\mathbf{P}\alpha_d, \mathbf{F}\alpha_d\} \subseteq SS$ or (ii) $\mathbf{P}\alpha_d \in SS$ and $Pre(\alpha_d)$ is false in S_t **then return** $\perp$.
end if
if $\{\alpha_d \mid \mathbf{Do}\alpha_d \in SS\}$ does not satisfy DC **then return** $\perp$.
end if
$SS'_d := SS$ // Start with an initial empty set.
repeat
    $SS''_d := SS'_d$.
    for each ground rule r in $\mathcal{N}_d$ **do**
        Let r be $SA_d \leftarrow \chi$ & $SA_{d,1}$ & $\ldots$ & $SA_{d,n}$.
        if χ is true in $S_d(t)$ **and** $\{SA_{d,1}, \ldots, SA_{d,n}\} \subseteq SS'_d$ **then**
            Add SA_d to SS'_d.
            if $SA_d = \mathbf{O}\alpha_d$ **then**
                Add $\mathbf{P}\alpha_d$ and $\mathbf{Do}\alpha_d$ to SS'_d.
            else if $SA_d = \mathbf{Do}\alpha_d$ **then**
                Add $\mathbf{P}\alpha_d$ to SS'_d.
            end if
            if there exists α_d s.t. **(i)** $\{\mathbf{P}\alpha_d, \mathbf{F}\alpha_d\} \subseteq SS'_d$ **or (ii)** $\mathbf{P}\alpha_d \in SS'_d$ **and** $Pre(\alpha_d)$ is false in $S_d(t)$ **then return** $\perp$ // Contradiction or invalid precondition.
            end if
            if $\{\alpha_d \mid \mathbf{Do}\alpha_d \in SS'_d\}$ does not satisfy DC **then return** $\perp$ // Denial constraint violated.
            end if
        end if
    end for
until $SS'_d = SS''_d$
return SS'_d

Algorithm 12 Ethical Status Set Computation Algorithm for Drone d

Input: Status set SS_{HQ}, state $S_d(t)$, norms $\mathcal{N}_d$, integrity constraints IC, action constraints AC, a function $\mathsf{conc}(\cdot)$, drone d's action space $\mathcal{A}_d(t)$, and an integer τ (the threshold for enumerating feasible sets).

Output: A set of feasible status sets $\{SS_d\}$ or $\bot$.

$DC \leftarrow \{$denial constraints in $AC\}$.
$LSS_d \leftarrow LSS(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$.
if $LSS_d = \bot$ **then**
  $LSS_d \leftarrow LSS(\emptyset, S_d(t), \mathcal{N}_d, DC)$.
  if $LSS_d = \bot$ **then return** $\bot$ // No ethically compliant status set exists.
  else
    $U = SS_{HQ}$.
  end if
else
  $U = SA(\mathcal{A}_d(t))$.
end if
$\overline{A}_d := \{\, \alpha_d \mid \alpha_d \in \mathcal{A}_d(t),\ Pre(\alpha_d)$ is false in $S_d(t)$ or $\mathbf{F}\alpha_d \in LSS_d\}$.
$\overline{SA}_d := \bigcup_{\alpha_d \in \overline{A}_d} \{\mathbf{Do}\alpha_d, \mathbf{O}\alpha_d, \mathbf{P}\alpha_d\}$.
$SA_d := U \setminus \big(\overline{SA}_d \cup LSS_d\big)$.
SA_d-$Do := \{\mathbf{Do}\alpha_d \mid \mathbf{Do}\alpha_d \in SA_d\}$.
SA_d-$FPO := SA_d \setminus \big(SA_d$-$Do\big)$.
$ToInspect := \{\, LSS_d \cup X \mid X \subseteq SA_d$-$FPO\}$. $Result := \emptyset$.
while $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
  $Candidates := ToInspect$. $ToInspect := \emptyset$.
  if some elements of $Candidates$ are feasible under IC & AC **then** // Feasibility check.
    for each feasible set $FeasSet_d$ in $Candidates$ **do**
      Add $FeasSet_d$ to $Result$.
      if $|Result| = \tau$ **then return** $Result$.
      end if
    end for
  else
    for each $Cand_d$ in $Candidates$ **do** // Expand candidates.
      for each $\mathbf{Do}\alpha_d \in \big(SA_d$-$Do \setminus Cand_d\big)$ **do**
        if $(Cand_d \cup \{\mathbf{Do}\alpha_d\}) \notin ToInspect$ **then**
          Add $(Cand_d \cup \{\mathbf{Do}\alpha_d\})$ to $ToInspect$.
        end if
      end for
    end for
  end if
end while
return $Result$

execute at this time step. Formally, let $\mathcal{F}_d$ be the set of all ethically feasible status sets for drone d. For each $SS_d \in \mathcal{F}_d$, the the CAS

$$X_{SS_d} := \{\alpha_d \mid \mathbf{Do}\alpha_d \in SS_d\}$$

is the **actual set of actions** that drone d will execute. All other subsets of actions are *masked out* by the constraints and operating rules. Thus, drone d only picks from these *ethically compliant* CASs, ensuring that every course of action is consistent with the prescribed ethical and legal norms.

C.2.7. Rewards for Drones

The primary reward formulations are presented in Section 5.4.3. This appendix provides the complete notation reference and additional details on the attack probability model.

C.2.7.1. Notation for Reward Functions. The following notation is used throughout the reward formulations:

- $\mathcal{D}_{\text{RED}}(t)$: Set of alive RED drones at time t
- $\mathcal{D}_{\text{BLUE}}(t)$: Set of alive BLUE drones at time t
- $\Delta B_d(t)$: Battery consumed by drone d during time step t
- κ_d: Ammunition cost coefficient for drone d
- σ_d: Survival bonus coefficient for drone d
- $v_c(t)$ or $v_{i,j}(t)$: Value/utility of cell c (or cell (i, j)) at time t
- $\mathcal{C}_{\text{danger}}(d', t)$: Set of cells in grid G that RED drone d' can target given its current payload $p_{d'}(t)$ and battery $B_{d'}(t)$
- $P_{\text{attack}}(d', c', t)$: Estimated probability that d' attacks cell c'

- $\|\mathbf{x}_d(t) - \mathbf{x}_c\|$: Euclidean distance between drone d and cell c at time t

C.2.7.2. Immediate Reward for BLUE Drone d at time t.

$$r_t^d = \alpha \cdot \left(\sum_{\substack{d' \in \mathcal{D}_{\text{RED}}(t) \\ killed(d,d',t)}} \text{cost}_{d'} \right) - \beta \cdot \Delta B_d(t) - \zeta \cdot fired(d,t) \cdot \kappa_d$$
$$+ \delta \cdot surv(d,t) \cdot \sigma_d + \rho \cdot \left(\sum_{c:resp(d,c,t)} alive(c,t) \cdot v_c(t) \right)$$
$$- \phi \cdot \sum_{c' \in \mathcal{C}_{\text{danger}}(d',t)} v_{c'}(t) \cdot P_{\text{attack}}(d', c', t)$$

C.2.7.3. Immediate Reward for RED Drone d' at time t.

$$r_t^{d'} = \alpha \cdot \left(\sum_{\substack{d \in \mathcal{D}_{\text{BLUE}}(t) \\ killed(d',d,t)}} \text{cost}_d \right) - \beta \cdot \Delta B_{d'}(t)$$
$$- \zeta \cdot fired(d',t) \cdot \kappa_{d'} + \delta \cdot surv(d',t) \cdot \sigma_{d'}$$
$$+ \rho \cdot \left(\sum_{c:resp(d',c,t)} alive(c,t) \cdot v_c(t) \right)$$
$$+ \phi \cdot \sum_{c' \in \mathcal{C}_{\text{danger}}(d',t)} v_{c'}(t) \cdot P_{\text{attack}}(d', c', t)$$

Note that this is not the same as the negative of the reward for a BLUE drone because the RED drones reward depends on the positions of all relevant BLUE drones that could threaten it, not just one.

C.2.7.4. Attack Probability Model. The probability that RED drone d' attacks cell c' within its remaining operational time is modeled as:

$$P_{\text{attack}}(d', c', t) = P_0(d', c', t) \cdot \left(1 - e^{-\mu \cdot B_{d'}(t)}\right)$$

where $\mu > 0$ is an attack urgency parameter and $P_0(d', c', t)$ is the base targeting probability.

When $\Delta t = 0$, the probability of attack is zero since the drone has no time to act. As Δt increases, the probability rises smoothly but with diminishing returns, and in the limit $\Delta t \to \infty$, the probability converges to $P_0(d', c', t)$.

C.2.7.5. Base Attack Probability Model.

$$P_0(d', c', t) = \frac{\left(\frac{v_{c'}(t)}{\|\mathbf{x}_{d'}(t) - \mathbf{x}_{c'}\| + \varepsilon}\right)^{\xi}}{\sum_{c'' \in \mathcal{C}_{\text{danger}}(d', t)} \left(\frac{v_{c''}(t)}{\|\mathbf{x}_{d'}(t) - \mathbf{x}_{c''}\| + \varepsilon}\right)^{\xi}}$$

where ε is a small constant to avoid division by zero (e.g., $\varepsilon = 10^{-3}$), and ξ (sharpness) controls selectivity.

This model produces a normalized probability distribution similar to a softmax. If two targets have the same value, the closer one is more likely to be chosen. If two targets are at the same distance, the higher-valued one is favored. In general, ξ controls how strongly the model discriminates between alternatives.

C.2.7.6. Team Reward. The team-level reward aggregates individual drone rewards:

$$R_t^{\text{BLUE}} = \sum_{d \in \mathcal{D}_{\text{BLUE}}(t)} r_t^d, \qquad R_t^{\text{RED}} = \sum_{d' \in \mathcal{D}_{\text{RED}}(t)} r_t^{d'}$$

C.3. Solving the Ethics-Guided GUARDIAN MDPs

The primary algorithms for Ethics-Guided Q-Learning (Algorithm 9) and HQ QMIX Coordination (Algorithm 10) are presented in Section 5.4.1.1. This appendix provides the complete pseudocode with additional implementation details.

C.3.1. Independent Q-Learning for Drones with Action Masking

For completeness, we reproduce the drone Q-learning algorithm from Section 5.4.1.1 with additional implementation details.

Algorithm 13 Drone d: Ethics-Guided Q-Learning (Detailed)

Input: (1) Q-network $Q_d(s, X)$ with parameters θ_d, (2) discount factor γ, (3) learning rate η, (4) exploration rate ϵ, (5) replay buffer $\mathcal{B}$, (6) FSS enumeration threshold τ, (7) norms $\mathcal{N}_d$, integrity constraints IC, action constraints AC.

for each episode or time step **do**
  Observe current state $s_d(t)$
  Compute feasible status sets: $\mathcal{F}_d \leftarrow$ Algorithm 12$\big(s_d(t), \mathcal{N}_d, IC, AC, \tau\big)$
  if $\mathcal{F}_d = \perp$ **then**
    (Fallback) If no feasible actions, do nothing or safe maneuver
    Continue to next time step
  end if
  Masked actions: $\widehat{\mathcal{A}}_d(s_d(t)) \leftarrow \{ X_{SS_d} \mid SS_d \in \mathcal{F}_d \}$.
  With probability ϵ, sample X uniformly from $\widehat{\mathcal{A}}_d(s_d(t))$;
otherwise pick $X = \arg\max_{X' \in \widehat{\mathcal{A}}_d(s_d(t))} Q_d\big(s_d(t), X'\big)$
  Execute CAS X, observe reward r_d and next state $s_d(t+1)$
  Store transition $(s_d(t),\ X,\ r_d,\ s_d(t+1))$ in buffer $\mathcal{B}$
  Update Q: sample minibatch from $\mathcal{B}$;
for each $(s,\ X,\ r,\ s')$ in the minibatch:

$$y = r + \gamma \max_{X' \in \widehat{\mathcal{A}}_d(s')} Q_d(s', X'; \theta_d)$$

$$L(\theta_d) = \big(y - Q_d(s, X; \theta_d)\big)^2$$

$$\theta_d \leftarrow \theta_d - \eta \nabla_{\theta_d} L(\theta_d)$$

end for

Algorithm 13 outlines the essential steps. Before selecting an action, drone d runs the *Ethical Status Set Computation Algorithm* (Algorithm 12 in Appendix C.2.6) to obtain $\mathcal{F}_d$, the feasible status sets. It then transforms them into CASs X_{SS_d}, forming the *masked* action space $\widehat{\mathcal{A}}_d(s)$. A standard Q-learning step is performed over these masked CASs. Because $\widehat{\mathcal{A}}_d(s)$ is pruned to only ethically compliant actions, the drone never attempts disallowed or forbidden maneuvers during training or execution.

C.3.2. QMIX for HQ Coordination

While individual drones learn local policies (Appendix C.3.1), the HQ aims to *coordinate* these drones to maximize team-level objectives. We adopt the QMIX algorithm, which is a popular *centralized training, decentralized execution* method.

HQ MDP Formulation. Let k be an HQ controlling drones $\{d_1, \ldots, d_m\}$. The HQ has an MDP $\mathcal{M}_k^{\text{HQ}} = (\mathcal{S}_k^{\text{HQ}}, \mathcal{A}_k^{\text{HQ}}, P_k^{\text{HQ}}, R_k^{\text{HQ}}, \gamma)$, where:

- s_k^{HQ} is the global (or near-global) state from HQ k's perspective.
- $\boldsymbol{a} = (a_1, \ldots, a_m)$ is a *joint action*, where a_i could be a suggested CAS for drone d_i.
- $P_k^{\text{HQ}}(s', s, \boldsymbol{a})$ describes the state transition at the HQ level.
- $R_k^{\text{HQ}}(s, \boldsymbol{a})$ is the team-level reward, capturing overall mission objectives.

Mixing Network. In QMIX, each drone d_i maintains a local Q-function Q_{d_i} (like in Appendix C.3.1), while the HQ learns a *mixing network*:

$$Q_{\text{tot}}(s_k^{\text{HQ}}, \boldsymbol{a}) \;=\; f\Big(Q_{d_1}(s_{d_1}, a_1),\; \ldots,\; Q_{d_m}(s_{d_m}, a_m);\; s_k^{\text{HQ}}\Big),$$

where $f(\cdot)$ is trained to approximate the *team-level Q-function*. A *monotonicity* constraint ensures that maximizing each drone's local Q-value leads to maximizing Q_{tot}.

Training and Execution Flow. Algorithm 14 sketches the HQ's procedure:

(1) The HQ observes $s_k^{\text{HQ}}(t)$ and queries each drone d_i for $Q_{d_i}(s_{d_i}(t), \cdot)$.

(2) The HQ uses the mixing network f to compute $Q_{\text{tot}}(s_k^{\text{HQ}}(t), \boldsymbol{a})$ for joint actions $\boldsymbol{a}$ and picks $\boldsymbol{a}^{\text{HQ}}(t) = \arg\max_{\boldsymbol{a}} Q_{\text{tot}}(\ldots)$.

(3) HQ k suggests $(a_1, \ldots, a_m)$ to each drone. *However*, each drone d_i will verify if a_i (or the CAS it implies) is in its *feasible set*. If it is not, d_i will default to a locally feasible CAS.

(4) The HQ collects the team reward $r_k(t)$ and next state $s_k^{\text{HQ}}(t+1)$, then updates its mixing network via temporal-difference learning.

This design ensures *ethical compliance* is preserved at the drone level, while the HQ pursues a higher-level global objective. When the HQ selects drone actions, it attempts to pick $(a_d)_{d \in \mathcal{D}_k}$ that jointly maximize Q_{tot}. However, each drone d *still* enforces its own feasibility mask. If the HQ suggests an infeasible CAS (e.g., a direct violation of deontic rules), the drone's local logic rejects or modifies it. Consequently, the HQ cannot force a drone to violate ethics; rather, it focuses on coordinating feasible CASs across the team to achieve higher-level goals. For completeness, we reproduce the HQ QMIX algorithm from Section 5.4.2 with additional context.

Here, we emphasize the fact that as each drone's RL policy is restricted to CASs that pass the deontic logic checks (Appendix C.2.6), we ensure that *no* unethical or forbidden behavior is ever attempted, *even* during exploratory phases. This mitigates risk in safety-critical domains.

Algorithm 14 HQ k: QMIX Coordination Algorithm (Detailed)

Input: (1) mixing network parameters θ, (2) discount factor γ, (3) learning rate η_{HQ}, (4) replay buffer $\mathcal{B}_{\text{HQ}}$, (5) monotonic constraint on $f(\cdot)$.

for each episode or time step **do**
  Observe HQ state $s_k^{\text{HQ}}(t)$
  for each drone $d_i \in \mathcal{D}_k$ **do**
    Obtain local Q-values $Q_{d_i}\big(s_{d_i}(t), \cdot\big)$
  end for
  Use $f(\cdot)$ to compute $Q_{\text{tot}}\big(s_k^{\text{HQ}}(t), \boldsymbol{a}\big)$ for candidate $\boldsymbol{a}$
  $\boldsymbol{a}^{\text{HQ}}(t) = \arg\max_{\boldsymbol{a}} Q_{\text{tot}}\big(s_k^{\text{HQ}}(t), \boldsymbol{a}\big)$
  HQ k suggests $a_i^{\text{HQ}}(t)$ to each drone d_i
  Drone feasibility check: each d_i confirms or replaces $a_i^{\text{HQ}}(t)$ based on its feasible CASs
  Execute final joint action $\boldsymbol{a}(t)$ on environment
  Observe $r_k(t)$ and next state $s_k^{\text{HQ}}(t+1)$
  Store $(s_k^{\text{HQ}}(t), \boldsymbol{a}(t), r_k(t), s_k^{\text{HQ}}(t+1))$ in $\mathcal{B}_{\text{HQ}}$
  Train mixing network: sample minibatch from $\mathcal{B}_{\text{HQ}}$

$$y_{\text{tot}} = r_k + \gamma \max_{\boldsymbol{a}'} Q_{\text{tot}}\big(s_k^{\text{HQ}}(t+1), \boldsymbol{a}'; \theta^-\big)$$

$$L_{\text{HQ}}(\theta) = \big(y_{\text{tot}} - Q_{\text{tot}}(s_k^{\text{HQ}}(t), \boldsymbol{a}(t); \theta)\big)^2$$

$$\theta \;\leftarrow\; \theta - \eta_{\text{HQ}}\, \nabla_\theta\, L_{\text{HQ}}(\theta)$$

end for

Although concurrency of actions can lead to an exponentially large search space (up to $2^{|\mathcal{A}_d|}$ subsets), the normative constraints and integrity rules prune this space significantly. Consequently, drones only deal with a tractable subset of CASs in practice.

Similarly, The HQ uses team-level RL (here, QMIX) to coordinate multiple drones. Crucially, a drone's local deontic logic always has the final say on whether a suggested CAS is admissible. Thus, HQ commands cannot violate ethical constraints, preserving overall system compliance.

The same ethics-guided principle applies whether we train drones independently (with or without a global HQ) or in a fully centralized multi-agent RL setting. As long as

action masking is enforced at the drone level, the resulting learned policies remain norm-compliant. Hence, this architecture ensures that *no* agent (drone or HQ) can inadvertently produce norm-violating behaviors at runtime while still leveraging off-the-shelf RL algorithms to learn policies in a complex multi-agent environment.

This layered approach guarantees ethical/legal compliance *by design*, reducing the risk of unwanted or forbidden actions in complex, multi-agent environments, e.g., GUARDIAN.

C.4. Assumptions in GUARDIAN Testbed

In developing the GUARDIAN testbed environment and formulating the mathematical models for the city grid, drones, CCTV cameras, and HQs, several assumptions have been made to simplify the implementation and focus on key aspects of the simulation. They are outlined in this section.

C.4.1. City Grid Assumptions

(1) **Grid Structure**:

- The city is represented as a two-dimensional grid G of fixed dimensions $M \times N$, where $M, N \in \mathbb{N}$.
- The grid consists of discrete cells located at integer coordinates (i, j), with $1 \leq i \leq M$ and $1 \leq j \leq N$.

(2) **Cell Values**:

- Each cell (i, j) has an initial value $v_{i,j}(0) \in \mathbb{R}_{>0}$, representing its importance in the grid.

- The initial cell values are assigned randomly using a uniform distribution:

$$v_{i,j}(0) \sim \text{Uniform}(v_{\min}, v_{\max})$$

where $v_{\min}, v_{\max} \in \mathbb{R}_{>0}$.

(3) **Cell Destruction**:

- Cells can be destroyed by RED Team drones, resulting in their value dropping to zero.
- When a cell is destroyed, it is considered *dead* and cannot be targeted again. But the drones can traverse through the cell.
- For simplicity, the change in cell value upon destruction is:

$$\Delta v_{i,j}(t) = -v_{i,j}(t)$$

resulting in $v_{i,j}(t+1) = 0$.

- CCTV in the cell is destroyed as well. Hence, $\text{alive}_c(t+1) = \text{false}$ if the CCTV c is located in the cell $\mathbf{x}_c = (i, j)$

(4) **Grid Dynamics**:

- The grid's structure (dimensions and cell positions) remains static throughout the simulation.
- Dynamic changes occur only in cell values due to destruction by drones; no other environmental factors alter cell values.
- We assume that the distance metric $\|\cdot\|$ stays consistent for one episode of the game. While we use the Chebyshev distance (maximum of absolute differences in coordinates) for grid environments, other metrics like Euclidean

or Manhattan distances can also be applied depending on the environment's characteristics. The choice of distance metric affects the drone's observation capabilities and can be adjusted based on specific scenario requirements.

C.4.2. Drone Assumptions

(1) **Team Composition**:

- Both the BLUE Team (team = 1) and the RED Team (team = 2) deploy multiple drones.

(2) **Initial Deployment**:

- Drones are randomly placed on the grid at positions that are unoccupied by other drones.

(3) **Drone Capabilities**:

- Each drone has battery capacity $B_d(0) \in \mathbb{R}_{>0}$ and payload $p_d(0) \in \mathbb{N}_0$.
- Drones have a fixed view range $r_d \in \mathbb{N}$ and fire range $f_d \in \mathbb{N}$. In our environment, we assume no obstacles are present. We assume homogeneous r_d and f_d values across drones of the same team for simplicity.
- BLUE Team drones do not fire at cells to avoid damaging the city; they only engage enemy drones.
- RED Team drones can fire at both enemy drones and cells.

(4) **Actions**:

- Actions have no duration, they are immediately executed at time t and their effect is there at time $t + 1$.

- Drones can move to adjacent cells (up, down, left, right) if the target cell is within the grid and not occupied by another drone.
- Drones can move onto destroyed cells.
- There is no friendly fire. Drones do not attack other drones or CCTVs from their own team.

(5) **Communication with HQ**:

- Drones prioritize orders from their HQ over their own decisions when communication is consistent.
- Communication failures may occur, in which case drones act autonomously based on their own MDPs.
- The possibility and frequency of communication failures are assumed and modeled in the simulation.

(6) **Observation Limitations**:

- Drones observe the environment within their view range r_d but have no knowledge beyond that.
- Observations are limited to the public states of other drones and cameras; drones cannot access others' internal states.

C.4.3. CCTV Camera Assumptions

(1) **Deployment**:

- CCTV cameras are deployed only by the BLUE Team (team = 1).
- The number of CCTV cameras $C_1 \geq 0$ is determined at the start of the simulation.

- Cameras are placed at fixed positions on the grid and do not move throughout the simulation.

(2) **Capabilities**:

- Each CCTV camera has an initial health $h_c(0) \in \mathbb{R}_{>0}$ and a view range $r_c \in \mathbb{N}$.
- Cameras do not have any offensive capabilities; they cannot attack or interfere with drones.
- The primary function is surveillance, providing observations to the BLUE Team's HQ.
- We assume that each CCTV, as long as it is not destroyed, sends constantly footage to the HQ.

(3) **Vulnerabilities**:

- A camera is destroyed if the corresponding cell where the camera is located is destroyed.

(4) **Observation Limitations**:

- Cameras observe the environment within their view range r_c.
- Observations are limited to public states; cameras cannot access internal states of drones.

C.4.4. Headquarters Assumptions

(1) **Control and Communication**:

- The HQ has the authority to assign actions to its drones (by sending them order).

- Communication between the HQ and drones may experience failures (probabilistic), leading to drones acting autonomously.

(2) **Observation and Decision-Making**:

- The HQ aggregates observations from its drones and CCTV cameras (for the BLUE Team) to form a global view.
- The HQ operates under its own MDP, making strategic decisions to optimize team objectives.
- Hence, we assume that the action HQ suggests to drone is more oriented toward team-specific objective than the drone's individual benefit.

(3) **Team Reward**:

- The HQ's reward function considers the cumulative rewards of its drones

C.4.5. Learning and Decision Making Assumptions

(1) **Drones' Learning Mechanism**:

- Drones use Independent Q-Learning (IQL) to learn policies based on their individual MDPs.
- Each drone updates its Q-values independently, without explicit coordination with other drones.

(2) **HQ's Learning Mechanism**:

- The HQ employs the QMIX algorithm to learn a joint policy for its drones, decomposing the team value function into individual value functions.
- The HQ assumes access to the individual Q-values of its drones for the mixing network in QMIX.

C.5. Additional Performance Metrics

We also measured the following additional performance evaluation metrics.

- *Payload efficiency.* Ratio of eliminated enemy drones to the total ammunition expended. Measures the efficiency of ammunition usage by drones, reflecting the accuracy and effectiveness in resource management.
- *Action entropy.* Average entropy of drones' action selection distributions. Represents how varied drone actions are over time—higher entropy indicates exploration, while lower entropy indicates consistency or exploitation of specific strategies.
- *Mean Q-values.* Average confidence level drones have in their learned action strategies.

Table C.1 shows that payload efficiency, action entropy, and mean Q-values are generally improved by compliance requirements—action entropy never worsens, and payload efficiency and mean Q-values only worsen in 3 cases each.

C.6. Full Experimental Results

Figures C.1–C.6 report the full experimental results.

C.7. Impact of Norm Combinations on Performance

The previous experiments utilize either the complete set of eight norms or no norms. To understand how specific norm subsets contribute to system performance, we conducted experiments with carefully selected norm combinations. These experiments maintain a

Payload efficiency (higher is better)

	16 BLUE drones		32 BLUE drones		64 BLUE drones	
	64x64	**128x128**	**64x64**	**128x128**	**64x64**	**128x128**
1:1	0.959	0.872	1.181	1.157	1.852	1.862
2:1	1.012	1.001	1.261	1.325	1.864	1.845
3:1	1.010	1.033	1.316	1.347	1.921	1.975
1:2	0.959	1.153	1.568	1.829	2.459	2.283
1:3	1.170	1.292	2.062	2.015	2.297	3.093

Action entropy (lower is better)

	16 BLUE drones		32 BLUE drones		64 BLUE drones	
	64x64	**128x128**	**64x64**	**128x128**	**64x64**	**128x128**
1:1	0.698	0.748	0.607	0.629	0.457	0.501
2:1	0.694	0.759	0.596	0.619	0.460	0.494
3:1	0.683	0.748	0.567	0.623	0.459	0.500
1:2	0.640	0.634	0.467	0.502	0.362	0.381
1:3	0.557	0.571	0.402	0.462	0.292	0.327

Mean Q-values (higher is better)

	16 BLUE drones		32 BLUE drones		64 BLUE drones	
	64x64	**128x128**	**64x64**	**128x128**	**64x64**	**128x128**
1:1	1.428	0.651	0.934	1.639	1.104	2.354
2:1	1.838	0.975	1.296	1.474	1.211	2.720
3:1	1.869	1.458	1.574	1.483	1.137	2.279
1:2	1.338	1.592	1.324	2.335	1.879	3.402
1:3	1.530	1.364	1.574	1.965	2.894	3.179

Table C.1. Compliance cost when varying *B:R* ratio, grid size, and number of BLUE drones.

1:1 BLUE-to-RED drone ratio across all configurations, ensuring symmetric competitive scenario.

C.7.0.1. Norm Selection for Experiment Design. We evaluated norm combinations of three different sizes: 2 norms (5 combinations), 4 norms (3 combinations), and 6 norms (2 combinations). Each combination was strategically selected to test specific hypotheses about norm interactions and their operational impact.

2-Norm Combinations:

- **[1, 2]**: Civilian area prohibition and friendly fire prohibition. Tests pure constraint-based protection without engagement guidance.

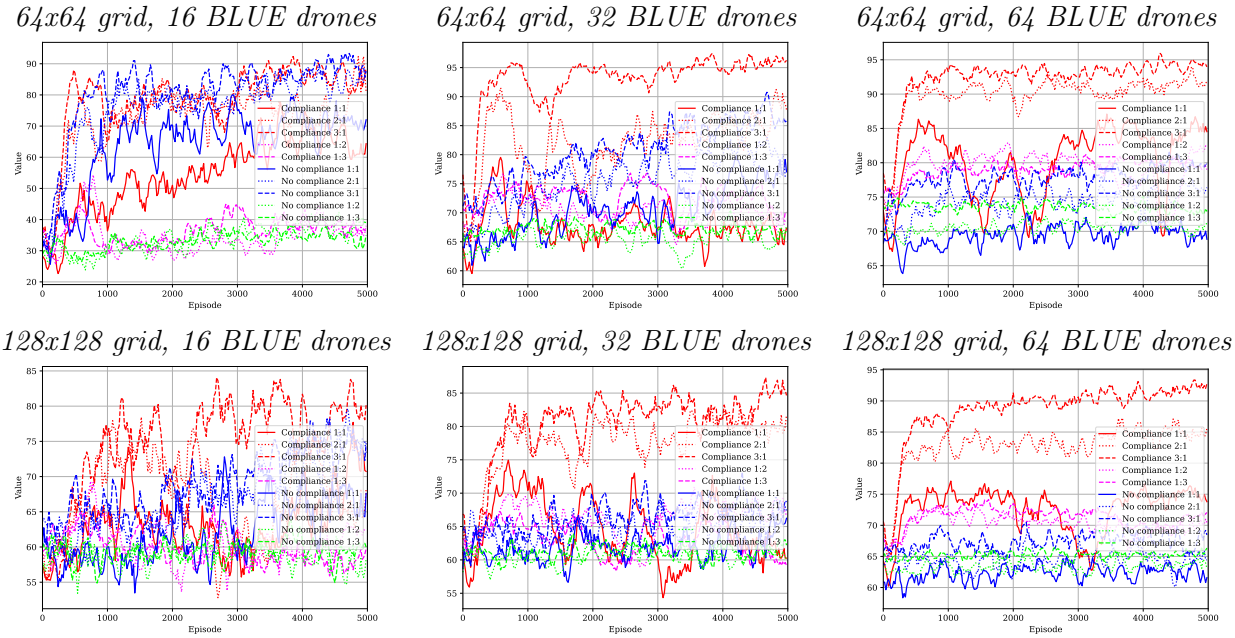


Figure C.1. City protection.

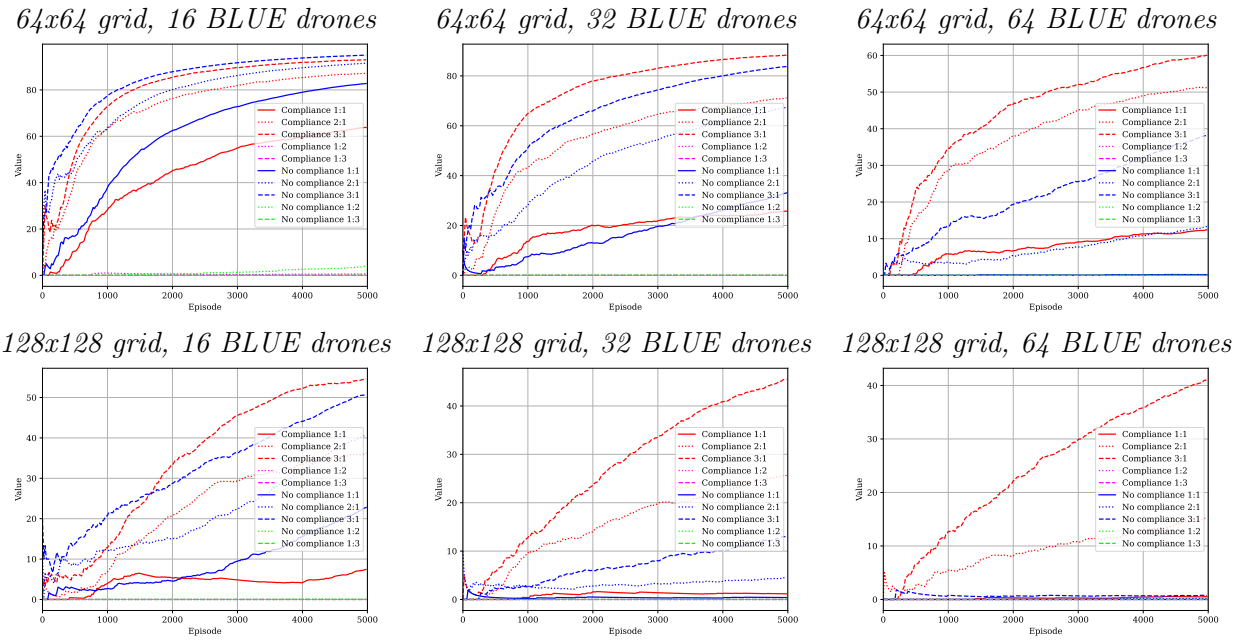


Figure C.2. Win rate.

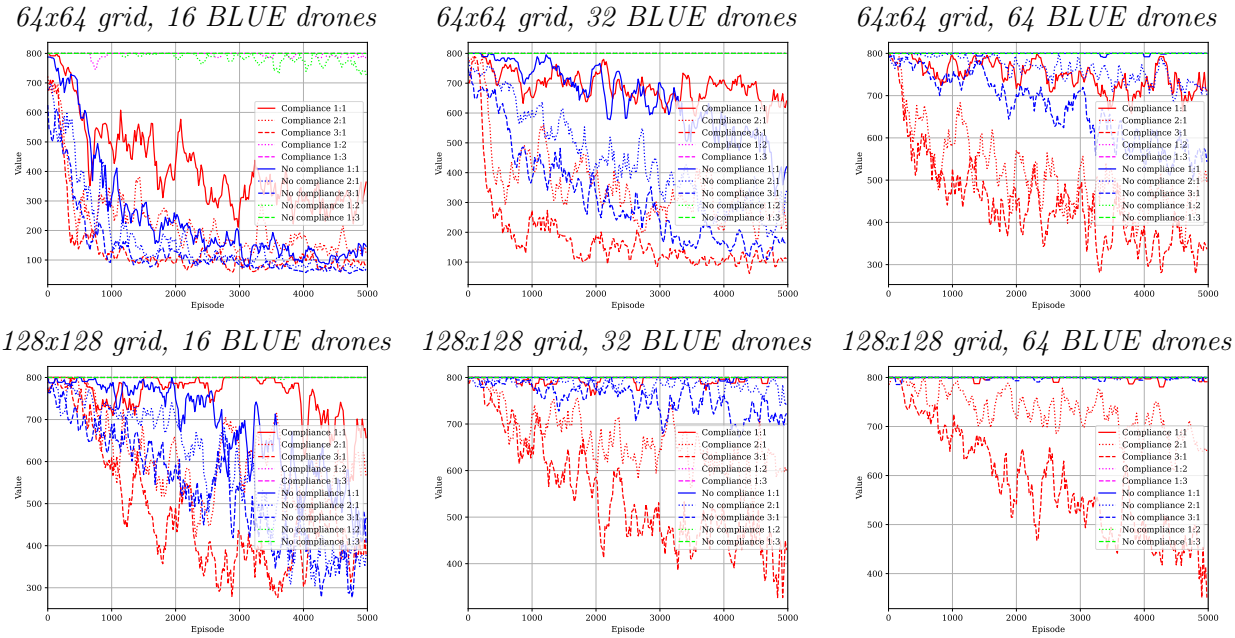


Figure C.3. Threat neutralization steps.

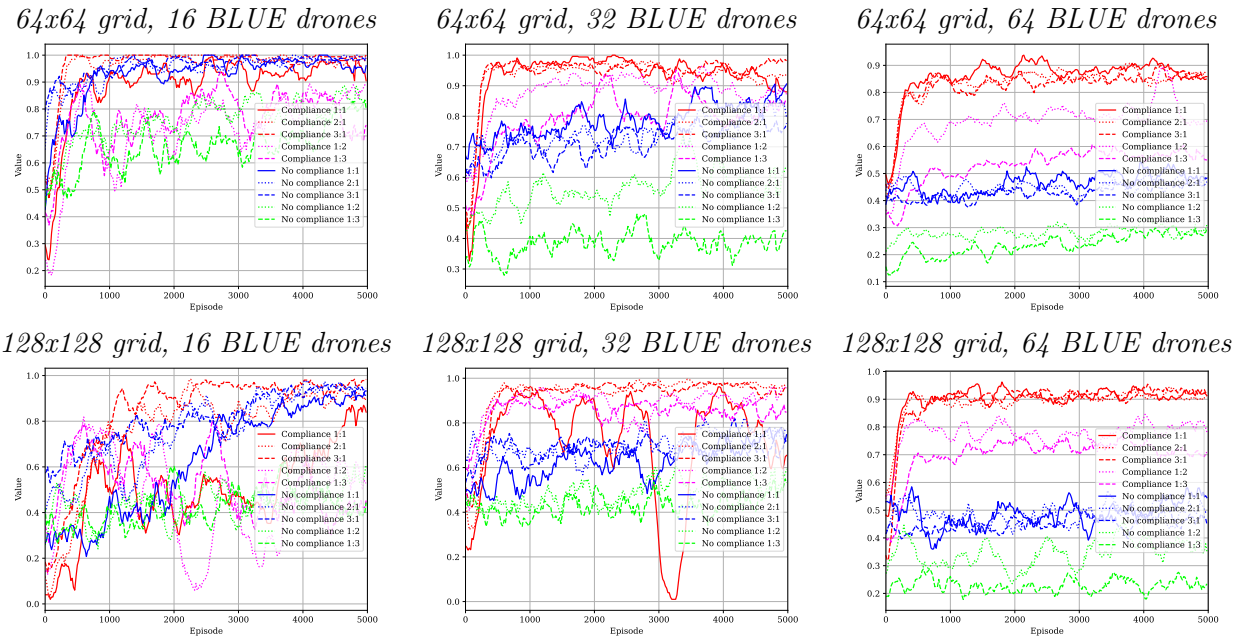


Figure C.4. Payload efficiency.

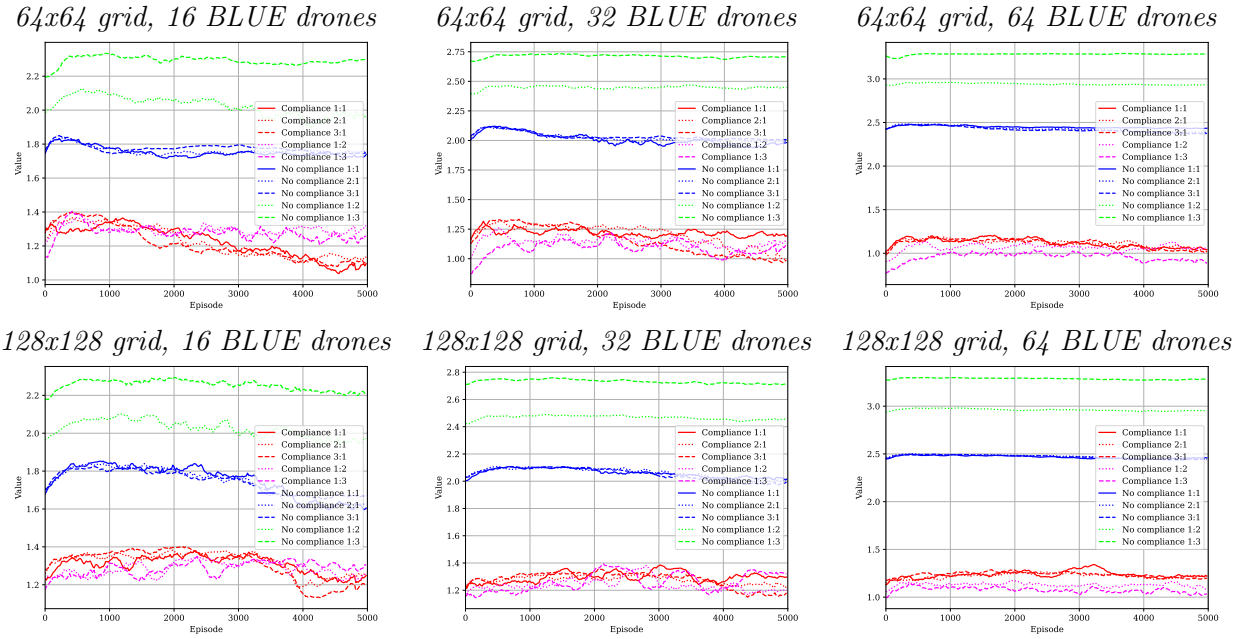


Figure C.5. Action entropy.

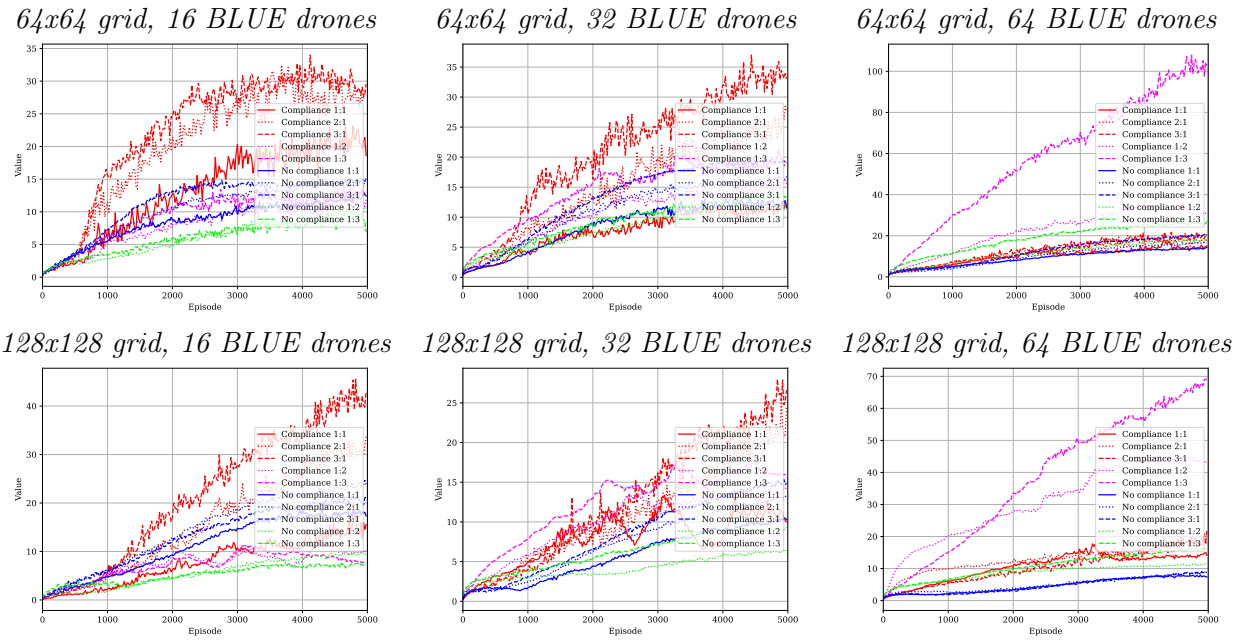
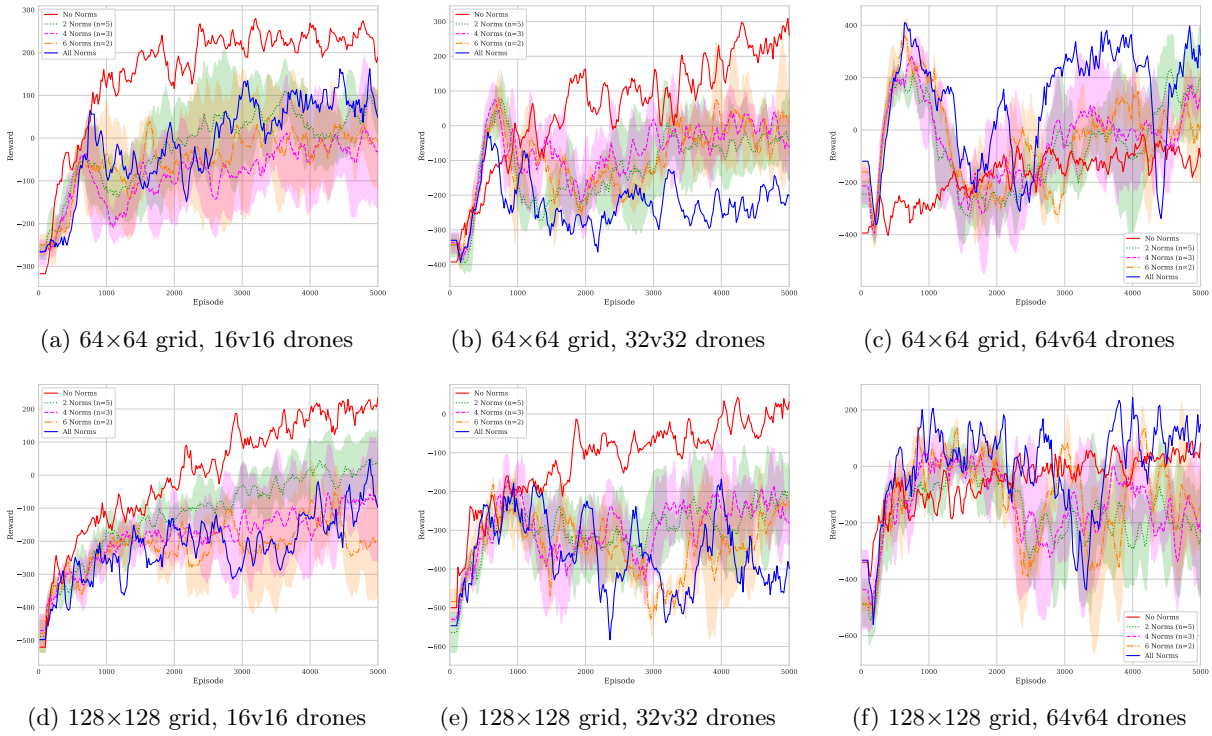


Figure C.6. Mean Q-values.



(a) 64×64 grid, 16v16 drones (b) 64×64 grid, 32v32 drones (c) 64×64 grid, 64v64 drones

(d) 128×128 grid, 16v16 drones (e) 128×128 grid, 32v32 drones (f) 128×128 grid, 64v64 drones

Figure C.7. Test rewards across norm combinations in symmetric (1:1) competitive scenarios. Lines represent mean performance with standard deviation bands.

- **[2, 3]**: Friendly fire prohibition and movement permissions. Evaluates mobility with minimal firing constraints.
- **[4, 5]**: Non-civilian firing permission and civilian area engagement permission. Examines permission-only frameworks without prohibitions.
- **[6, 7]**: Threat-based firing prohibition and high-value neighbor obligation. Tests the interaction between strategic restraint and mandatory engagement.
- **[7, 8]**: High-value neighbor obligation and extremely high-value neighbor obligation. Evaluates redundant obligation structures.

4-Norm Combinations:

- **[1, 2, 3, 4]**: Core prohibitions (1-2) with basic permissions (3-4). Tests whether fundamental constraints and permissions suffice for coordination.
- **[2, 5, 6, 7]**: Engagement rules and threat assessment norms. Examines combat-focused norms without civilian protection.
- **[5, 6, 7, 8]**: Advanced threat and utility-based norms. Tests high-level strategic norms without basic constraints.

6-Norm Combinations:

- **[1, 2, 3, 4, 5, 6]**: All basic norms plus civilian engagement rules. Excludes only the obligation norms (7-8).
- **[3, 4, 5, 6, 7, 8]**: All permissions and advanced engagement norms. Excludes basic prohibitions (1-2).

C.7.0.2. Results. Figure C.7 reveals compelling patterns across deployment densities. In sparse scenarios (16v16 drones), no-norms achieves highest rewards while complete norm sets underperform. Notably, this relationship inverts at high density (64v64 drones), where the complete 8-norm set dominates all configurations.

The 2-norm combinations exhibit particularly poor performance across most scenarios. Combination [1,2] (pure prohibitions) paralyzes the decision-making by constraining actions without providing guidance. Combination [4,5] (pure permissions) creates confusion among drones through uncoordinated engagement. The [7,8] pairing of obligations fails because both norms trigger simultaneously without supporting permissions, violating feasibility conditions.

The 4-norm combinations show intermediate performance. Configuration [1,2,3,4] provides basic functionality but lacks engagement obligations critical for high-density coordination. Configuration [5,6,7,8] contains sophisticated engagement logic but missing basic prohibitions (1-2) allows friendly fire and civilian area violations.

The 6-norm combinations reveal the importance of complete deontic chains. Configuration [1,2,3,4,5,6] includes all constraints and permissions but lacks obligations (7-8), resulting in passive behavior during critical engagements. Configuration [3,4,5,6,7,8] has obligations and permissions but missing prohibitions (1-2) undermines ethical constraints.

The failure of partial norm sets results in the feasible status set computation. From Section 3.3, the LSS algorithm requires:

$$\text{If } \mathbf{O}\,\alpha \in \text{SS}_d \text{ then } \mathbf{P}\,\alpha \in \text{SS}_d$$

Consider combination [7,8] under high-density scenario. When $\text{AllNeighborsAbove}(i, j, t, \lambda)$ holds, Norm 7 generates: $\mathbf{O}\,\text{FireAtDrone}_d(r)$. Hence, LSS closure requires: $\mathbf{P}\,\text{FireAtDrone}_d(r)$. However, without Norm 5, this permission is absent. Hence, the status set becomes infeasible: $\text{SS}_d \notin \mathcal{F}_d(s)$

Similarly, combination [4,5] provides permissions without prohibitions. The action space:

$$\mathcal{A}_d(s) = \{X_{\text{SS}_d} \mid \text{SS}_d \in \mathcal{F}_d(s)\}$$

becomes overcrowded with permitted but uncoordinated actions, leading to targeting collisions.

Also, the transition from sparse to dense deployments fundamentally changes coordination requirements. At 64v64 drones in a 64×64 grid, the probability that multiple BLUE drones observe the same RED target approaches unity. Without obligations (Norms 7-8), each drone independently maximizes its Q-function:

$$a_i = \arg\max_{a \in \mathcal{A}_{d_i}} Q(s, a)$$

This leads to redundant targeting where multiple drones engage the same enemy while others remain unengaged. The complete norm set prevents this through obligation-driven coordination. When Norm 7 triggers for drone d_i, it creates a deterministic assignment that other drones respect, preventing collisions.