Responsible Defense from Multi-Drone Attacks

PhD Final Defense

Tonmoay Deb
Department of Computer Science
Northwestern University

Committee Members:
Dr. V.S. Subrahmanian, Computer Science (Advisor and Chair)
Dr. Larry Birnbaum, Computer Science
Dr. Nabil Alshurafa, Computer Science
Dr. Alberto Quattrini Li, Department of Computer Science, Dartmouth College

December 8, 2025

Outline

Upcoming Section

The Emerging Drone Threat

CNN World Africa Americas Asia Australia China Europe India More Watch Listen Subscribe Sign In

WORLD › EUROPE • 3 MIN READ

Investigations launched after more unexplained drone sightings off two European coasts

DEC 6, 2025

By Tim Lister and Billy Stockwell

A general view of the Dublin city skyline in 2017. *(Stephen McCarthy/Sportsfile/Getty Images)*

The Emerging Drone Threat

Drone proliferation creates security challenges

- Terror groups actively use drones (ISIS, Hezbollah, PKK)
- State-sponsored warfare (Ukraine, India-Pakistan 2025)
- Critical infrastructure attacks (Saudi oil refineries, 2019)

Urban environments face complex airspaces

- Legitimate vs. malicious drone flights
- Real-time threat assessment required
- Legal and ethical constraints on responses

Notable Incidents

• Gatwick Airport (2018):
1,000+ flights canceled

• Saudi Aramco Attack (2019):
$2B damage to oil facilities

• Jammu Air Force (2021):
First drone attack on Indian military

• Ukraine Conflict (2022-ongoing):
Thousands of drone strikes on cities

• India-Pakistan (2025):
Drone warfare escalation

Central Research Question

Overarching Challenge

How can autonomous systems defend **regions** from drone attacks while maintaining **legal and ethical compliance**?

Four Core Research Problems:

1. **Early Threat Detection**: Distinguish threats from benign flights within seconds
2. **Data Scarcity**: Generate realistic threat trajectories when real data is scarce
3. **Legal Reasoning**: Make decisions that satisfy all applicable legal/ethical constraints
4. **Adaptive Defense**: Learn effective strategies while maintaining strict compliance

This dissertation makes significant progress towards each of these four problems.

Problem 1: Early Threat Prediction (DEWS)

Drone Threat Prediction Problem (DTPP)

Given the **first** j **seconds** of a live trajectory and drone metadata, classify whether the flight will become threatening.

Motivation:

- Security officials need actionable intelligence in 30 seconds
- Limited observation data for decision-making
- Must balance false positives (disrupting legitimate flights) vs. false negatives (missing threats)

Our Solution: DEWS—Drone Early Warning System

- First system for early drone threat prediction from partial trajectories
- Achieves $F_1 > 0.80$ within 30 seconds of observation
- Validated on real data from The Hague with Dutch police

Problem 2: Threat-Conditioned Trajectory Generation (STATE)

Data Scarcity Problem

Real threat trajectory data is scarce. How can we generate realistic synthetic data for training and testing defense systems?

Motivation:

- Few real-world threat trajectories exist
- Privacy and security constraints limit data sharing
- Need diverse scenarios across different geographic regions

Our Solution: STATE—Safe and Threatening Adversarial Trajectory Engine

- cGAN-based architecture for threat-conditioned trajectory synthesis
- Generates realistic trajectories over unseen regions
- 35.8% improvement in F1-score vs. baselines (expert validated)

Problem 3 & 4: Compliant Defense (GUARDIAN)

The Compliance Gap

Standard RL maximizes reward without considering legal norms. How can defenders act **legally** while learning to be **effective**?

Motivation:

- Must reason about complex deontic rules (obligations, permissions, prohibitions)
- Need to ensure *all* actions are compliant before execution
- Learn effective policies within the compliant action space

Our Solution: GUARDIAN—Governance-Unified Aerial Reinforcement-Defense

- Uses deontic logic to specify drone warfare legal/ethical constraints
- Leverages feasible status set computation algorithms from prior work
- Integrates FSS-based action masking with multi-agent RL
- Counterintuitive finding: Compliance can *improve* **defensive** performance

Upcoming Section

DEWS: Problem Formulation

Drone Threat Prediction Problem (DTPP)

Given the **first** j **seconds** of a live trajectory τ and drone metadata, *classify* whether the flight will become threatening.

Formal Definition:

- Trajectory: $\tau_d = ((\ell_1^d, t_1), \ldots, (\ell_n^d, t_n))$
- Temporal restriction: $tr(\tau_d, j)$
- Threat score: $y(\tau_d) \in [1, 10]$
- Learn: $f_{lev} : (d, tr(\tau_d, j)) \to \{0, 1\}$, where $f(d, tr(\tau_d, j)) = 1$ if threat $\geq lev$

Observation Windows:

$j \in \{1, 5, 10, 20, 30, 60, 180, 360\}$ seconds

Threat Thresholds:

- Low: score < 4 (61% of data)
- Medium: score $\in [4, 8)$ (27%)
- High: score ≥ 8 (12%)

Key Research Focus:

Analyze earliness vs. accuracy tradeoff

DEWS: Real-World Dataset (The Hague)

Dataset (provided by Dutch Police):

- **8 months**: Dec 2020 – Jul 2021
- **349 trajectories** from The Hague
- Senhive RF sensors (25 km radius)
- **18 distinct drones** observed
- Avg. flight: 5 min / 750 m traveled

Expert Annotation:

- Dutch police & municipality experts
- Inter-annotator agreement: $\kappa = 0.772$ (substantial)
- Anonymized for public release

Dataset Statistics by Threat Level

Metric	Low	Med	High
Trajectories	213	94	42
Duration (s)	265	298	286
Distance (m)	435	988	752
Altitude (m)	63	116	101
Speed (km/h)	7.1	14.6	10.4

First real-world drone threat dataset

DEWS: System Overview

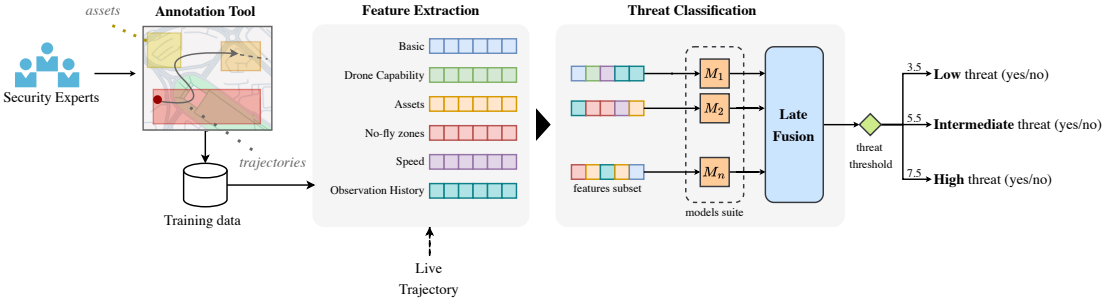


Figure: DEWS Architecture. Data set preparation involves annotating *asset values* and *drone trajectories* by *police*. Subsequently, DEWS extracts *features* and trains 11 classifiers $M_1, \cdots, M_{11}$ to yield 11 predictions which are integrated using *late fusion* to predict the final *threat 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.

DEWS: Feature Engineering

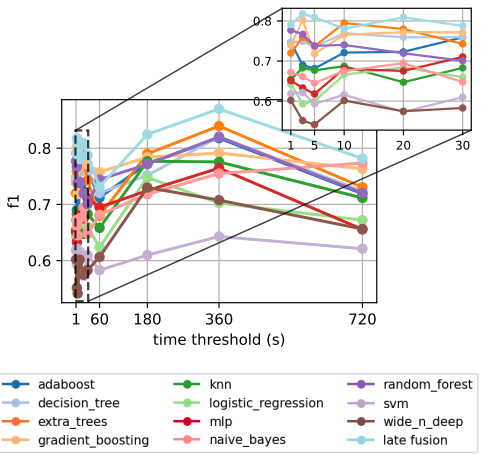
Six Feature Categories (Total: 110 features)

1. **Basic Flight Parameters**
   - Waypoint counts, duration
   - Spatial spread, bounding box
2. **Drone Capabilities**
   - Max payload, battery capacity
   - Top speed, range
3. **Altitude & Speed Metrics**
   - Min/mean/max/percentiles
   - Ground speed profiles
4. **No-Fly Zone (NFZ) Compliance**
   - Intrusion flags
   - Min distance to restricted areas
   - % time in NFZ
5. **Asset-Value Features**
   - Max/mean/cumulative values
   - Ground asset importance
6. **Historical Similarity**
   - self and cross-similarity with past trajectories

DEWS: Performance Results

Key Findings

1. **Rapid Early Warning**
   - $F_1 > 0.80$ after 30 seconds
   - Precision $> 90\%$, Recall $\sim 75\%$
   - Peaks at $F_1 = 0.96$ (6-min prefix)
2. **Late Fusion Superiority**
   - Consistently beats all 11 base models
   - Robust across observation windows
3. **Key Feature Insight**
   - Asset-value features are #1 predictor
   - *Where* a drone flies matters
4. **Operational Efficiency**
   - 3-second end-to-end latency
   - Actionable time buffer for response



High-Threat F_1 score vs. observation window

DEWS: Summary

Contributions:

- First system for early drone threat prediction from partial trajectories
- Novel integration of geospatial asset values into threat assessment
- Ensemble approach achieving 0.80+ F1 within 30 seconds

Practical Impact:

- Enables proactive defense planning
- Reduces operator cognitive load
- Provides actionable time buffer for counter-measures

Limitation: Single-city dataset (The Hague) $\rightarrow$ **Motivates STATE**

Upcoming Section

STATE: Problem Formulation

Trajectory Representation:

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

Threat-Conditioned Trajectory Generation

Learn a generative model $\mathcal{G}$ such that:

$$\mathcal{G} : (\mathcal{A}, \hat{\theta}, z) \to \tau$$

where $\mathcal{A}$ is any geographical region, $\hat{\theta} \in \{0, 1\}$ is the target threat class, and z is a latent noise vector.

Key Objective: Generate trajectories over *unseen* regions while preserving threat-specific behavioral patterns learned from available data.

STATE: Data Representation Module

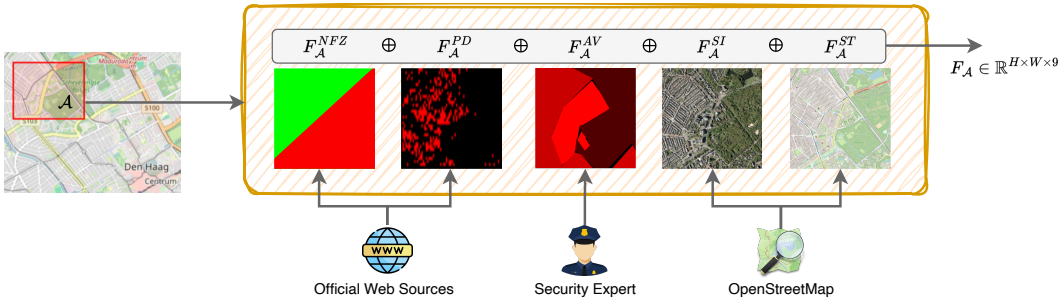


Figure: The target geographical region $\mathcal{A}$ is represented with a multi-channel feature tensor, including the *No-Fly Zone* Map $F_{\mathcal{A}}^{NFZ}$, the *Population Density Map* $F_{\mathcal{A}}^{PD}$, the *Satellite Imagery* $F_{\mathcal{A}}^{SI}$, the *Street Map* $F_{\mathcal{A}}^{ST}$, and the *Asset Value Map* $F_{\mathcal{A}}^{AV}$.

STATE: Waypoint Generator

Waypoint Generator $\mathcal{G}$:

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

Architecture:

- CLIP-based encoder: $F \in \mathbb{R}^{H \times W \times 9} \rightarrow X_F \in \mathbb{R}^{D_h}$
- Threat encoder: $\hat{\theta} \rightarrow X_{\hat{\theta}} \in \mathbb{R}^{D_h}$
- Latent noise: $z \sim \mathcal{N}(0, \mathbf{I}_d) \rightarrow X_z \in \mathbb{R}^{D_h}$
- Concatenation: $X' = \left[X_F \oplus X_{\hat{\theta}} \oplus X_z\right] \in \mathbb{R}^{3 \cdot D_h}$
- Decoder: Transposed conv layers $\rightarrow \hat{\tau} \in \{0,1\}^{128 \times 128}$

Trajectory Validity Discriminator $\mathcal{M}^V : (\hat{\tau}, F, \theta) \rightarrow p_{\hat{\tau}}^V \in [0,1]$

- Distinguishes real from synthetic trajectories
- Ensures generated trajectories match real data distribution

STATE: Threat Alignment Network and Training

Pre-trained classifier $\mathcal{M}^T$ ensures threat consistency (based on DEWS)

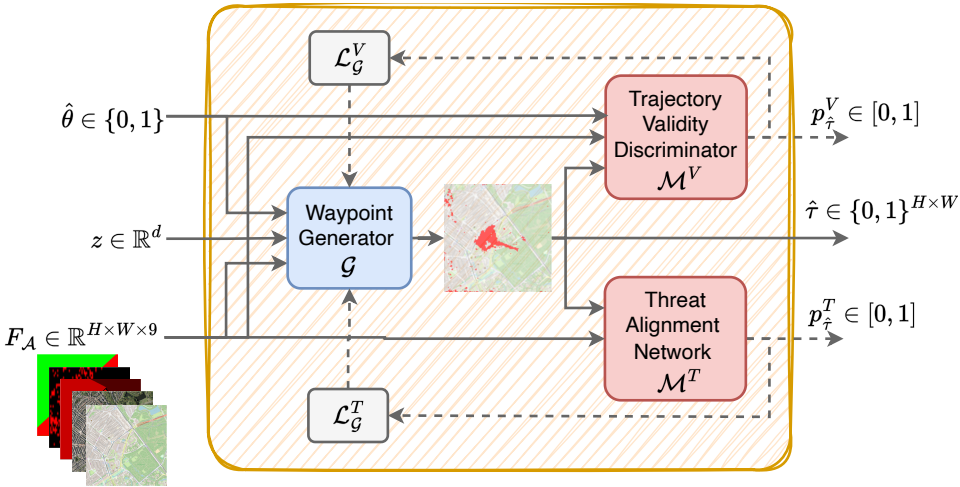
$$\mathcal{M}^T : (\hat{\tau}, F) \rightarrow p_{\hat{\tau}}^T = P(\hat{\theta}|\hat{\tau}, F) \in [0, 1]$$

- Pre-trained threat classifier
- Ensures generated trajectories match target threat label
- Fixed during generator training

Training Strategy:

- $\mathcal{M}^V$ is trained adversarially with $\mathcal{G}$
- $\mathcal{M}^T$ **fixed** during GAN training (not adversarial)
- Provides auxiliary loss: $\mathcal{L}_\mathcal{G} = \lambda_V \mathcal{L}_{\mathcal{M}^V} + \lambda_T \mathcal{L}_{\mathcal{M}^T}$

STATE: Adversarial Training Dynamics



STATE: Trajectory Reconstruction Module

Challenge: Convert unordered planar projection $\hat{\tau}$ to temporal trajectory τ

Temporal Sequencing Process:

1. Identify contour Ω of largest connected component in $\hat{\tau}$

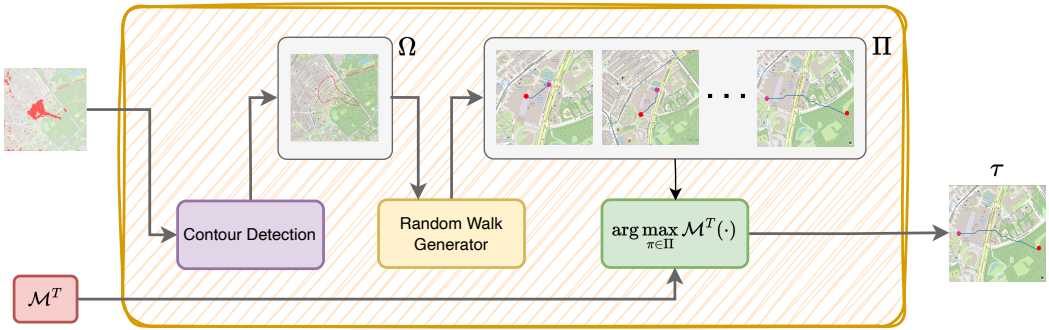
$$\Omega = \{w_1, w_2, \ldots, w_l\}$$

2. For all waypoint pairs $(w_s, w_e) \in \Omega$ with $d(w_s, w_e) < \xi$:
   - Generate candidate trajectory π via stochastic random walk
   - Creates set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_L\}$ where $L = \binom{l}{2}$
3. Select best trajectory using $\mathcal{M}^T$:

$$\tau^* = \arg\max_{\pi \in \Pi} \mathcal{M}^T(\pi, F)$$

Altitude Assignment: Conditional on threat class $\hat{\theta}$ (drawn from the learned distribution)

STATE: Trajectory Reconstruction Visualization



STATE: Comparison with Baselines

Evaluation Metrics (lower is better):

- **MDE** (Mean Distance Error): spatial accuracy
- **SSIM** (Structural Similarity): trajectory diversity
- **JSD-AV**: asset value distribution similarity
- **JSD-TL**: trajectory length distribution similarity

Method	MDE ↓		SSIM ↓		JSD-AV ↓		JSD-TL ↓	
	Threat	Safe	Threat	Safe	Threat	Safe	Threat	Safe
Random Walk	17.38	15.04	0.928	0.953	0.0065	0.0042	0.054	0.025
Monte Carlo	15.40	14.42	0.944	0.960	0.0051	0.0061	0.050	0.023
LSTM	5.25	3.19	0.882	0.907	0.0045	0.0052	0.027	0.017
VAE	9.61	10.71	0.978	0.940	0.0025	0.0038	0.043	0.019
Traj-GAN	8.29	6.52	0.856	0.826	0.0040	0.0043	0.028	0.017
STATE	**1.27**	**1.62**	**0.661**	0.664	**0.0010**	0.0020	0.015	**0.005**

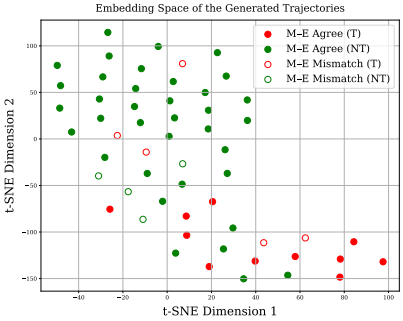
Table: Lower is better. STATE achieves **75.8%** improvement over LSTM on threatening trajectories.

STATE: Expert Evaluation on Unseen Regions

Setup:

- Generated 200 trajectories over *unseen* regions
- 100 per model (STATE, VAE), 50 safe + 50 threatening
- Two Dutch police officers independently annotated

Method	F1-Score $\theta = 0$	F1-Score $\theta = 1$	Acc.
VAE	0.857	0.522	0.780
STATE	**0.888**	**0.709**	**0.839**
Δ	+3.6%	**+35.8%**	+7.6%



t-SNE of STATE embeddings. Filled = model-expert agreement; Hollow = disagreement.

Key Result: STATE *generalizes* to unseen regions with **35.8% F1 improvement** on threatening trajectories vs. VAE

STATE: Contributions and Limitations

Contributions:

- Novel cGAN architecture for threat-conditioned trajectory synthesis
- Dual feedback: realism + threat alignment
- Outperforms 5 baseline methods
- Addresses data scarcity for rare threats
- Enables testing in unmonitored areas

Limitations:

- Single-city evaluation (The Hague)
- Binary threat classification
- Relies on pre-trained threat classifier
- Dual-use considerations

Next Challenge: Given threat detection, how should defenders *respond*? $\rightarrow$ **GUARDIAN**

Upcoming Section

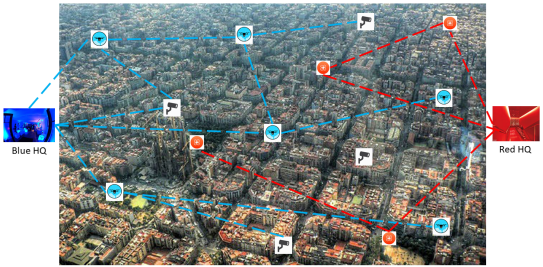
GUARDIAN: The Core Challenge

The Compliance Gap in RL:

- Standard RL: $\pi^* = \arg\max_\pi \mathbb{E}[\text{reward}]$
- Legal/ethical norms $\mathcal{N}$ are *external*
- High-reward actions may violate laws

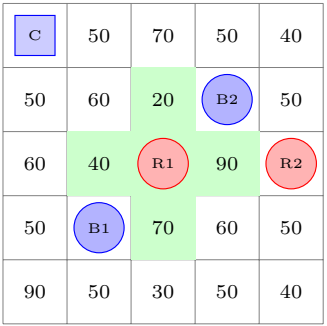
Real-World Challenge:

- BLUE team (defenders) must comply with norms
- RED team (attackers) ignores norms
- **Question:** Does compliance disadvantage BLUE?



BLUE vs. RED scenario over Paris

GUARDIAN: Motivating Example



Legend:

- Blue drone
- Red drone
- CCTV
- Civilian area

Cell values: infrastructure importance

Scenario

- 5×5 urban grid
- Green cells: civilian areas
- Values: infrastructure importance
- R1: Red drone in high-value cell
- B1, B2: Blue defenders

Ethical Constraints

- Norm 1: No firing in civilian areas unless immediate threat
- Norm 2: Obligatory to fire when high-value neighbors at risk
- ⇒ Multiple feasible actions

GUARDIAN Foundation: Deontic Logic Framework

Deontic Operators specify normative status of actions:

$\mathbf{P}\alpha$	Action α is *permitted*
$\mathbf{O}\alpha$	Action α is *obligatory* (must do)
$\mathbf{F}\alpha$	Action α is *forbidden*
$\mathbf{Do}\alpha$	Action α *will be executed*

Deontic Rules encode legal/ethical constraints:

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

where χ is a conjunction of state atoms

We'll see concrete examples on the next slide.

GUARDIAN: Deontic Rules

Example Rules for BLUE Drones:

1. **Never fire at cells:**
   F $FireAtCell_d(i, j) \leftarrow Blue(d)$
2. **Prohibit friendly fire:**
   F $FireAtDrone_d(d') \leftarrow Blue(d) \wedge Blue(d') \wedge SameTeam(d, d')$
3. **Prohibit firing in civilian areas (unless immediate threat):**
   F $FireAtDrone_d(d') \leftarrow Blue(d) \wedge Red(d') \wedge CivilianArea(i, j) \wedge \neg ImmediateThreat(d')$
4. **Obligate engagement when neighbors are high-value:**
   O $FireAtDrone_d(d') \leftarrow$
   $Blue(d) \wedge Red(d') \wedge ImmediateThreat(d') \wedge AllNeighborsAbove(i, j, t, \lambda)$

8 total rules developed based on suggestions from security experts.

Status Sets

Status Set (SS): A set of ground status atoms specifying the deontic status of each action.

Example Status Set for drone d:

$$SS_d = \left\{ \begin{array}{l} \mathbf{P}\ \mathit{MoveTo}_d(3,4),\ \mathbf{Do}\ \mathit{MoveTo}_d(3,4), \\ \mathbf{P}\ \mathit{FireAtDrone}_d(d'),\ \mathbf{F}\ \mathit{FireAtCell}_d(3,4) \end{array} \right\}$$

Key Question: Is a status set *feasible*? That is, does it satisfy all deontic rules, constraints, and logical consistency requirements?

Feasible Status Set (FSS): Definition

Feasible Status Set

A status set SS_d is **feasible** if it satisfies all 8 conditions:

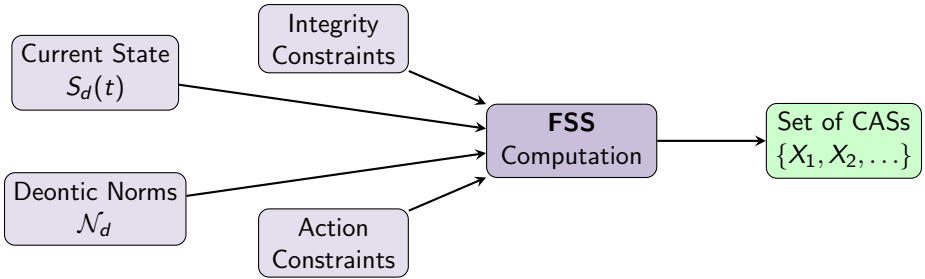
1. $\mathbf{O}\alpha \in SS_d \Rightarrow \mathbf{P}\alpha \in SS_d$ *(obligations imply permission)*
2. $\mathbf{O}\alpha \in SS_d \Rightarrow \mathbf{Do}\alpha \in SS_d$ *(obligations must be done)*
3. $\mathbf{Do}\alpha \in SS_d \Rightarrow \mathbf{P}\alpha \in SS_d$ *(done actions must be permitted)*
4. $\mathbf{P}\alpha \in SS_d \Rightarrow \mathbf{F}\alpha \notin SS_d$ *(no permission-prohibition conflict)*
5. $\mathbf{P}\alpha \in SS_d \Rightarrow$ preconditions of α satisfied *(physical feasibility)*
6. SS_d is closed under operating rules $\mathcal{N}_d$ *(rule closure)*
7. $\{\alpha \mid \mathbf{Do}\alpha \in SS_d\}$ satisfies action constraints AC *(action consistency)*
8. Resulting state satisfies integrity constraints IC *(state consistency)*

Key Output: The **Concurrent Action Set (CAS)**:

$$X_{SS_d} = \{\alpha \mid \mathbf{Do}\alpha \in SS_d\}$$

This is the set of actions drone d will actually execute.

From FSS to Dynamically Masked Action Space



Result: Each CAS is a **legally compliant** combination of actions.

Dynamically Masked Action Space:

$$\widehat{\mathcal{A}}_d(s) = \{X_{SS} \mid SS \in \mathcal{F}_d(s)\}$$

The drone can *only* select from these compliant action sets.

GUARDIAN: Integrating FSS with Reinforcement Learning

Use FSS computation to dynamically mask the RL action space.

RL with Dynamically Masked Actions

At each state s, the drone optimizes:

$$\pi_d^* = \arg\max_{\pi_d} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_d(s(t), X(t))\right]$$

Subject to: $X(t) \in \widehat{\mathcal{A}}_d(s(t))$ at every step.

Result:

- Infeasible FSSs are *never* explored during training
- Infeasible FSSs are *never* executed during deployment
- Learned policies are **compliant by construction**

GUARDIAN: Learning Architecture

Two-Level Hierarchy:

- **Drone Level:** Independent Q-Learning with action masking

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

- **HQ Level:** QMIX for centralized training, decentralized execution

$$Q_{\text{tot}}(s^{HQ}, \mathbf{a}) = f(Q_{d_1}, \ldots, Q_{d_m}; s^{HQ})$$

Key Property: Even HQ cannot override drone compliance.

- HQ suggests actions; drones verify feasibility via FSS
- If HQ suggestion violates norms, drone substitutes feasible alternative

GUARDIAN: Experimental Setup

Grid Configuration

- Grid sizes: 64×64, 128×128
- Cell values: $v_{i,j}(0) \in [0, 100]$
- 3 CCTV cameras (view range 10)

Drone Parameters

- Blue drones: 16, 32, 64
- Battery capacity: 100 units
- Payload: 3 units
- View range: 5, Fire range: 1

Blue:Red Ratios

- 1:1 (symmetric)
- 2:1, 3:1 (defender advantage)
- 1:2, 1:3 (attacker advantage)

Training

- Deep Independent Q-Learning
- QMIX for HQ coordination
- 5,000 episodes
- Stochastic communication (80% success)

Metrics: Reward, City Protection, Win Rate, Q Values, etc.

GUARDIAN: Reward

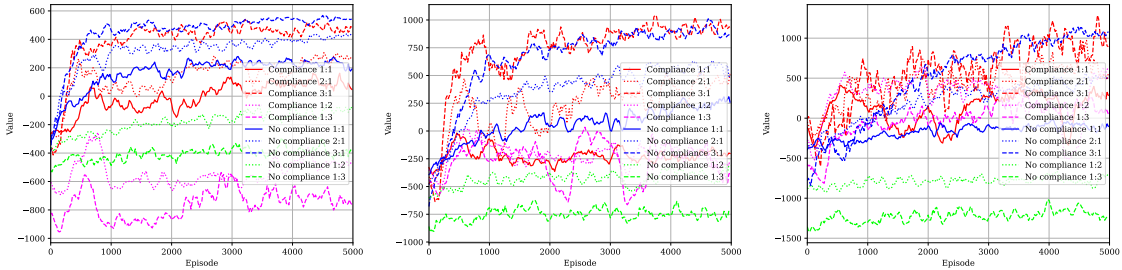


Figure: 64×64 grid with 16, 32, and 64 BLUE drones

Three Critical Observations:

(Obs 1) Compliance generally worsens performance (expected)

(Obs 2) On 64×64 grid with 32 and 64 BLUE drones, compliance *improves* performance when RED drones are majority (1:2, 1:3 ratios)

(Obs 3) Larger problems ⇒ compliance reward approaches or exceeds non-compliance

GUARDIAN: City Protection

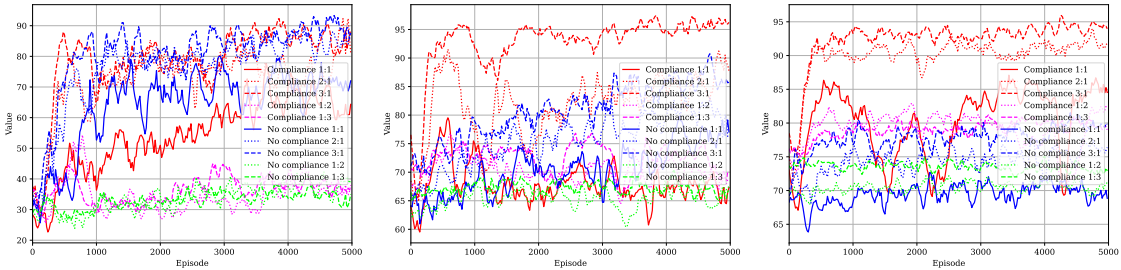


Figure: 64×64 grid: 16, 32, 64 BLUE drones.

Key Finding: Compliance *improves* city protection in most cases (up to 31.3% improvement). Only 4 of 30 configurations show degradation.

GUARDIAN: Compliance Cost Analysis

Compliance Cost: $CC = \frac{\text{Performance with Norms}}{\text{Performance without Norms}}$

B:R	16 BLUE 64×64	16 BLUE 128×128	32 BLUE 64×64	32 BLUE 128×128	64 BLUE 64×64	64 BLUE 128×128
City Protection (higher = better)						
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

Table: Compliance cost ratio: > 1 means compliance *improves* performance.

Key Findings:

- Compliance improves city protection in most cases (up to 20.9% improvement)
- Only 4 cases show degradation (up to 14%)
- Hence, deontic constraints often *help* rather than hurt defense

GUARDIAN: Computational Efficiency

Per-Step Decision Time (ms)

Drones	Comp	CAS	QMIX
16	Yes	215.6	23.7
16	No	0	15.3
32	Yes	446.8	26.2
32	No	0	23.6
64	Yes	554.9	70.8
64	No	0	41.6

Observations

- CAS computation: 2.6× increase (16→64 drones)
- QMIX inference: 3× increase
- Total: 625.7ms for 64 drones
- Real-time capable

Training Time

- 5,000 episodes, 64×64 grid, 64 drones
- With norms: ∼630 hours
- Without norms: ∼80 hours
- Overhead: 7.9× for training
- Acceptable for offline training

Key Takeaway

- Training overhead significant
- But inference remains real-time
- Practical for deployment
- Legal compliance worth the cost

GUARDIAN: Summary of Findings

Counterintuitive Result:
Legal compliance does *not* necessarily handicap defenders

Key Insights:

1. **Reduced Search Space:** Constraints focus exploration on viable policies
2. **Implicit Curriculum:** Deontic rules guide learning for complex problems
3. **Asymmetric Advantage:** RED faces full complexity; BLUE has structured search
4. **Scale-Dependent:** Benefits most pronounced at larger problem scales
5. **Practical Viability:** 625.7ms decision time enables real-time deployment

Broader Implications:

- Hard constraints can *facilitate* learning (not just constrain it)
- Formal compliance guarantees achievable without sacrificing effectiveness
- Challenges assumption that "tied hands" = tactical disadvantage

Upcoming Section

Concluding Remarks

Central Message:

Regional airspace can be defended proactively and responsibly through integration of prediction, legal reasoning, and learning.

Key Takeaway:

- Legal compliance is **not** a handicap
- Constraints can **facilitate** learning at scale
- Formal guarantees + effectiveness are achievable
- Interdisciplinary approach is essential

Vision:

Autonomous defense systems that are simultaneously effective, compliant, adaptive, transparent, and subject to meaningful human oversight

This dissertation makes significant progress towards this vision.

Publications

1. **Deb, T.**, De Laaf, S., La Gatta, V., Lemmens, O., Lindelauf, R., Van Meerten, M., Meerveld, H., Neeleman, A., Postiglione, M., and Subrahmanian, V.S. "A Drone Early Warning System (DEWS) for Predicting Threatening Trajectories." *IEEE Intelligent Systems*, 2025.
2. **Deb, T.**, Jeong, M., Molinaro, C., Pugliese, A., Quattrini Li, A., Santos, E., Subrahmanian, V.S., and Zhang, Y. "Declarative Logic-Based Pareto-Optimal Agent Decision Making." *IEEE Transactions on Cybernetics*, vol. 54, no. 12, pp. 7147–7162, 2024.
3. **Deb, T.**, Dix, J., Jeong, M., Molinaro, C., Pugliese, A., Quattrini Li, A., Santos Jr., E., Subrahmanian, V.S., Yang, S., and Zhang, Y. "DUCK: A Drone-Urban Cyber-Defense Framework Based on Pareto-Optimal Deontic Logic Agents." *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.
4. **Deb, T.**, Jeong, M., Molinaro, C., Pugliese, A., Quattrini Li, A., Santos Jr., E., Zhang, Y., and Subrahmanian, V.S. "GUARDIAN: Governance-Unified Aerial Reinforcement-Defense In Accordance with Norms." To be submitted.
5. **Deb, T.**, Denisenko, N., Gao, C., La Gatta, V., de Laaf, S., Neeleman, A., Sola, L., and Subrahmanian, V.S. "STATE: Safe and Threatening Adversarial Trajectory Engine." To be submitted.
6. Mutzari, D., **Deb, T.**, Molinaro, C., Pugliese, A., Subrahmanian, V.S., and Kraus, S. "Defending a City from Multi-Drone Attacks: A Sequential Stackelberg Security Games Approach." *Artificial Intelligence*, vol. 349, p. 104425, 2025. **(S2D2)**
7. Jeong, M., Molinaro, C., **Deb, T.**, Zhang, Y., Pugliese, A., Santos, E., Subrahmanian, V.S., and Quattrini Li, A. "Multi-Object Active Search and Tracking by Multiple Agents in Untrusted, Dynamically Changing Environments." *Autonomous Robots*, vol. 50, no. 1, 2026.

Acknowledgements

Advisor

Dr. V.S. Subrahmanian
Northwestern University

Committee Members

Dr. Larry Birnbaum
Dr. Nabil Alshurafa
Dr. Alberto Quattrini Li

Lab Members

Valerio La Gatta
Marco Postiglione
Saurabh Kumar
Lirika Sola
Natalia Denisenko
Chongyang Gao

Collaborators

Netherlands Police
(Sven de Laaf, Odette Lemmens)

Municipality of The Hague
(Max van Meerten, Iris Neeleman)

Dartmouth College
(Mingi Jeong, Eugene Santos Jr.)

University of Calabria
(Cristian Molinaro, Andrea Pugliese)

Chinese Academy of Science
(Youzhi Zhang)

Bar-Ilan University
(Dolev Mutzari, Sarit Kraus)

Special thanks to my wife and my parents for their unwavering support.

Thank You!

Questions & Discussion

Tonmoay Deb
tonmoay.deb@northwestern.edu

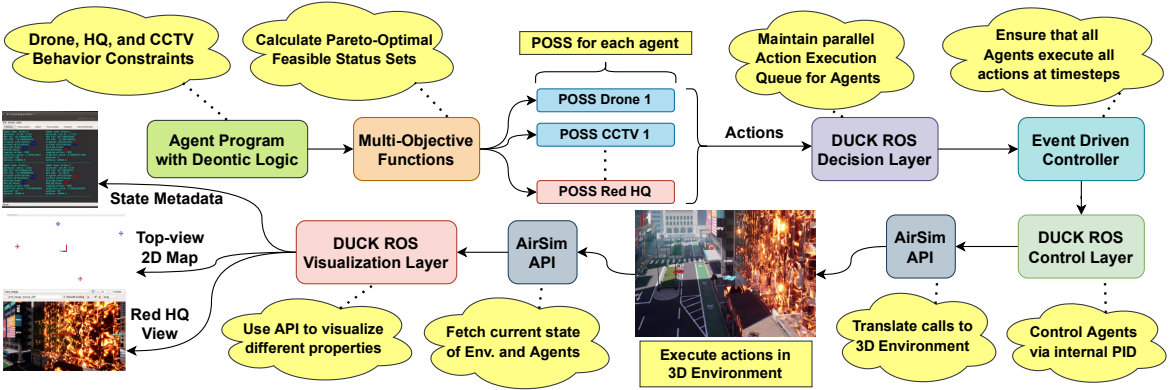
Department of Computer Science
Northwestern University

Backup Slides

Future: DUCK 3D Simulation Environment Testbed

Figure: Three-screen view: (L) Blue/RHQ commands, (C) Ground-truth Unreal rendering, (R) Sensor overlays + UI.

DUCK Testbed Architecture



Integrity Constraints (IC) & Action Constraints (AC)

Integrity constraints ensure system consistency and safety. They must hold in the resulting state.

IC_1: Engagement Within Firing Range

$$\leftarrow \ FireAtDrone_d(d') \ \wedge \ \neg InFireRange(d, d')$$

"Cannot fire at a drone that is out of range"

Action constraints define permissible combinations of concurrent actions within a single time step.

AC_1: Single Target Engagement

$$\leftarrow \ FireAtDrone_d(d_1) \ \wedge \ FireAtDrone_d(d_2) \ \wedge \ d_1 \neq d_2$$

"Cannot engage multiple targets simultaneously"

Computing Feasible Status Sets: LSS

Least Status Set (LSS) Algorithm: Computes the minimal status set that satisfies all deontic closure conditions.

Key Steps:

1. **Initialize:** Start with initial constraints (e.g., HQ orders)
2. **Enforce deontic closure:**
   - If $\mathbf{O}\alpha \in SS$: add $\mathbf{P}\alpha$ and $\mathbf{Do}\alpha$
   - If $\mathbf{Do}\alpha \in SS$: add $\mathbf{P}\alpha$
3. **Apply operating rules:** For each rule whose body is satisfied, add the head
4. **Check for contradictions:**
   - If both $\mathbf{P}\alpha$ and $\mathbf{F}\alpha$ exist: return $\bot$
   - If $\mathbf{P}\alpha$ but precondition false: return $\bot$
   - If denial constraints violated: return $\bot$
5. **Repeat** until fixpoint reached

Output: Minimal baseline status set, or $\bot$ if no compliant option exists.

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$,

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$, IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$, IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$, IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$

2. $LSS_d \leftarrow \text{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.
- **Line 2**: Compute *Least Status Set* via LSS (try with HQ orders first).

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$,
IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$
2. $LSS_d \leftarrow \text{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$
3. **if** $LSS_d = \perp$ **then** $LSS_d \leftarrow \text{LSS}(\emptyset, S_d(t), \mathcal{N}_d, DC)$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.
- **Line 2**: Compute *Least Status Set* via LSS (try with HQ orders first).
- **Line 3**: If HQ conflicts with norms, retry without HQ orders.

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$,
IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$
2. $LSS_d \leftarrow \text{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$
3. **if** $LSS_d = \perp$ **then** $LSS_d \leftarrow \text{LSS}(\emptyset, S_d(t), \mathcal{N}_d, DC)$
4. **if** $LSS_d = \perp$ **then return** $\perp$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.
- **Line 2**: Compute *Least Status Set* via LSS (try with HQ orders first).
- **Line 3**: If HQ conflicts with norms, retry without HQ orders.
- **Line 4**: If still $\perp$, no compliant option exists.

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$, IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$
2. $LSS_d \leftarrow \text{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$
3. **if** $LSS_d = \perp$ **then** $LSS_d \leftarrow \text{LSS}(\emptyset, S_d(t), \mathcal{N}_d, DC)$
4. **if** $LSS_d = \perp$ **then return** $\perp$
5. $\overline{A}_d \leftarrow \{\alpha_d \mid Pre(\alpha_d) \text{ false OR } \mathbf{F}\alpha_d \in LSS_d\}$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.
- **Line 2**: Compute *Least Status Set* via LSS (try with HQ orders first).
- **Line 3**: If HQ conflicts with norms, retry without HQ orders.
- **Line 4**: If still $\perp$, no compliant option exists.
- **Line 5**: Identify infeasible actions (preconditions fail or forbidden).

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$, IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$
2. $LSS_d \leftarrow \text{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$
3. **if** $LSS_d = \bot$ **then** $LSS_d \leftarrow \text{LSS}(\emptyset, S_d(t), \mathcal{N}_d, DC)$
4. **if** $LSS_d = \bot$ **then return** $\bot$
5. $\overline{A}_d \leftarrow \{\alpha_d \mid Pre(\alpha_d) \text{ false OR } \mathbf{F}\alpha_d \in LSS_d\}$
6. $SA_d \leftarrow SA(\mathcal{A}_d) \setminus (\overline{SA}_d \cup LSS_d)$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.
- **Line 2**: Compute *Least Status Set* via LSS (try with HQ orders first).
- **Line 3**: If HQ conflicts with norms, retry without HQ orders.
- **Line 4**: If still $\bot$, no compliant option exists.
- **Line 5**: Identify infeasible actions (preconditions fail or forbidden).

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$,
IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$
2. $LSS_d \leftarrow \text{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$
3. **if** $LSS_d = \perp$ **then** $LSS_d \leftarrow \text{LSS}(\emptyset, S_d(t), \mathcal{N}_d, DC)$
4. **if** $LSS_d = \perp$ **then return** $\perp$
5. $\overline{A}_d \leftarrow \{\alpha_d \mid Pre(\alpha_d) \text{ false OR } \mathbf{F}\alpha_d \in LSS_d\}$
6. $SA_d \leftarrow SA(\mathcal{A}_d) \setminus (\overline{SA}_d \cup LSS_d)$
7. $SA_d\text{-}Do \leftarrow \{\mathbf{Do}\alpha_d \mid \mathbf{Do}\alpha_d \in SA_d\}$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.
- **Line 2**: Compute *Least Status Set* via LSS (try with HQ orders first).
- **Line 3**: If HQ conflicts with norms, retry without HQ orders.
- **Line 4**: If still $\perp$, no compliant option exists.
- **Line 5**: Identify infeasible actions (preconditions fail or forbidden).
- **Lines 7-8**: **Key Step**: Separate **Do** atoms from **F**/**P**/**O** atoms.

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$, IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$
2. $LSS_d \leftarrow \text{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$
3. **if** $LSS_d = \bot$ **then** $LSS_d \leftarrow \text{LSS}(\emptyset, S_d(t), \mathcal{N}_d, DC)$
4. **if** $LSS_d = \bot$ **then return** $\bot$
5. $\overline{A}_d \leftarrow \{\alpha_d \mid Pre(\alpha_d) \text{ false OR } \mathbf{F}\alpha_d \in LSS_d\}$
6. $SA_d \leftarrow SA(\mathcal{A}_d) \setminus (\overline{SA}_d \cup LSS_d)$
7. $SA_d\text{-}Do \leftarrow \{\mathbf{Do}\alpha_d \mid \mathbf{Do}\alpha_d \in SA_d\}$
8. $SA_d\text{-}FPO \leftarrow SA_d \setminus SA_d\text{-}Do$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.
- **Line 2**: Compute *Least Status Set* via LSS (try with HQ orders first).
- **Line 3**: If HQ conflicts with norms, retry without HQ orders.
- **Line 4**: If still $\bot$, no compliant option exists.
- **Line 5**: Identify infeasible actions (preconditions fail or forbidden).
- **Lines 7-8**: **Key Step**: Separate **Do** atoms from **F**/**P**/**O** atoms.

Ethical Status Set Computation Algorithm: Initialization & Setup

Input: HQ orders SS_{HQ}, State $S_d(t)$, Norms $\mathcal{N}_d$, IC, AC, Actions $\mathcal{A}_d(t)$, Threshold τ

1. $DC \leftarrow \{\text{denial constraints in } AC\}$
2. $LSS_d \leftarrow \text{LSS}(SS_{HQ}, S_d(t), \mathcal{N}_d, DC)$
3. **if** $LSS_d = \perp$ **then** $LSS_d \leftarrow \text{LSS}(\emptyset, S_d(t), \mathcal{N}_d, DC)$
4. **if** $LSS_d = \perp$ **then return** $\perp$
5. $\overline{A}_d \leftarrow \{\alpha_d \mid Pre(\alpha_d) \text{ false OR } \mathbf{F}\alpha_d \in LSS_d\}$
6. $SA_d \leftarrow SA(\mathcal{A}_d) \setminus (\overline{SA}_d \cup LSS_d)$
7. $SA_d\text{-}Do \leftarrow \{\mathbf{Do}\alpha_d \mid \mathbf{Do}\alpha_d \in SA_d\}$
8. $SA_d\text{-}FPO \leftarrow SA_d \setminus SA_d\text{-}Do$
9. $ToInspect \leftarrow \{LSS_d \cup X \mid X \subseteq SA_d\text{-}FPO\}$; $Result \leftarrow \emptyset$

- **Lines 1-2**: Gather inputs: HQ orders, state, norms, constraints.
- **Line 2**: Compute *Least Status Set* via LSS (try with HQ orders first).
- **Line 3**: If HQ conflicts with norms, retry without HQ orders.
- **Line 4**: If still $\perp$, no compliant option exists.
- **Line 5**: Identify infeasible actions (preconditions fail or forbidden).
- **Lines 7-8**: **Key Step**: Separate **Do** atoms from **F**/**P**/**O** atoms.
- **Line 9**: Initialize BFS frontier with all FPO combinations.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. while $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**
14. Add $FeasSet_d$ to $Result$

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**
14. Add $FeasSet_d$ to $Result$
15. **if** $|Result| = \tau$ **then return** $Result$

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**
14. Add $FeasSet_d$ to $Result$
15. **if** $|Result| = \tau$ **then return** $Result$
16. **else**

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**
14. Add $FeasSet_d$ to $Result$
15. **if** $|Result| = \tau$ **then return** $Result$
16. **else**
17. **for each** $Cand_d$ in $Candidates$ **do**

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.
- **Lines 16-19**: **Key**: Expand by adding *only* **Do** atoms (not all status atoms).

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**
14. Add $FeasSet_d$ to $Result$
15. **if** $|Result| = \tau$ **then return** $Result$
16. **else**
17. **for each** $Cand_d$ in $Candidates$ **do**
18. **for each Do**$\alpha_d \in (SA_d\text{-}Do \setminus Cand_d)$ **do**

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.
- **Lines 16-19**: **Key**: Expand by adding *only* **Do** atoms (not all status atoms).

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**
14. Add $FeasSet_d$ to $Result$
15. **if** $|Result| = \tau$ **then return** $Result$
16. **else**
17. **for each** $Cand_d$ in $Candidates$ **do**
18. **for each** $\mathbf{Do}\alpha_d \in (SA_d\text{-}Do \setminus Cand_d)$ **do**
19. Add $(Cand_d \cup \{\mathbf{Do}\alpha_d\})$ to $ToInspect$

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.
- **Lines 16-19**: **Key**: Expand by adding *only* **Do** atoms (not all status atoms).
- **Line 19**: BFS: one **Do** atom per expansion step.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**
14. Add $FeasSet_d$ to $Result$
15. **if** $|Result| = \tau$ **then return** $Result$
16. **else**
17. **for each** $Cand_d$ in $Candidates$ **do**
18. **for each** $\mathbf{Do}\alpha_d \in (SA_d\text{-}Do \setminus Cand_d)$ **do**
19. Add $(Cand_d \cup \{\mathbf{Do}\alpha_d\})$ to $ToInspect$
20. **end while**

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.
- **Lines 16-19**: **Key**: Expand by adding *only* **Do** atoms (not all status atoms).
- **Line 19**: BFS: one **Do** atom per expansion step.

Ethical Status Set Computation Algorithm: BFS Expansion & Termination

10. **while** $ToInspect \neq \emptyset$ **and** $|Result| < \tau$ **do**
11. $Candidates \leftarrow ToInspect$; $ToInspect \leftarrow \emptyset$
12. **if** some $SS \in Candidates$ are feasible under IC & AC **then**
13. **for each** feasible $FeasSet_d$ in $Candidates$ **do**
14. Add $FeasSet_d$ to $Result$
15. **if** $|Result| = \tau$ **then return** $Result$
16. **else**
17. **for each** $Cand_d$ in $Candidates$ **do**
18. **for each** $\mathbf{Do}\alpha_d \in (SA_d\text{-}Do \setminus Cand_d)$ **do**
19. Add $(Cand_d \cup \{\mathbf{Do}\alpha_d\})$ to $ToInspect$
20. **end while**
21. **return** $Result$

- **Line 10-11**: Loop until no candidates OR collected τ feasible sets.
- **Line 12**: Check feasibility against IC and AC.
- **Lines 13-15**: Collect feasible sets into Result.
- **Lines 16-19**: **Key**: Expand by adding *only* **Do** atoms (not all status atoms).
- **Line 19**: BFS: one **Do** atom per expansion step.
- **Line 21**: Return all collected feasible status sets $\mathcal{F}_d = \{SS_1, SS_2, \ldots\}$ — set of feasible status sets.
  Each FSS yields a **Concurrent Action Set**: $X_{SS_i} = \{\alpha \mid \mathbf{Do}\alpha \in SS_i\}$

POSS: Proof Overview

Three Key Complexity Results in POSS Chapter:

1. **Proposition (Membership):** Deciding if a status set is a Pareto-optimal feasible status set is in co-NP
2. **Theorem (Hardness):** Deciding if a status set is a Pareto-optimal feasible status set is co-NP-hard
3. **Proposition (Closure):** The Closure algorithm runs in polynomial time

Why these results matter:

- Establishes computational complexity bounds for decision problems
- Justifies the need for heuristic algorithms in practice
- Provides theoretical foundation for GUARDIAN's use of POSS

POSS: Membership in co-NP (Proposition)

Claim: Deciding if a status set SS is Pareto-optimal feasible is in co-NP.

Proof Intuition (5 Steps):

1. **Feasibility Check:** Verify SS satisfies all 8 conditions of Definition (feasible status set) – this is polynomial time
2. **Complementary Problem:** "Is SS NOT Pareto-optimal?" is in NP because:
   - We can *guess* a witness status set SS' that dominates SS
   - Verify SS' is feasible (polynomial time)
   - Verify SS' dominates SS on objective functions (polynomial time)
3. **Verification:** Given witness SS', check $\forall f \in OF : f(SS') \geq f(SS)$ and $\exists f : f(SS') > f(SS)$
4. **Polynomial Verification:** All checks are polynomial in problem size
5. **Conclusion:** Since complement is in NP, original problem is in co-NP

POSS: co-NP-Hardness (Theorem) – Part 1

Claim: Deciding if a status set is Pareto-optimal feasible is co-NP-hard.

Proof Strategy: Reduction from 3-Colorability (known NP-complete)

Step 1: Problem Setup

- Given graph $G = (V, E)$
- Question: Can G be 3-colored? (adjacent vertices different colors)

Step 2: Reduction Construction

- **Actions:** For each vertex $v \in V$:
  - $\text{coloring}_a(v, c_1), \text{coloring}_a(v, c_2), \text{coloring}_a(v, c_3)$ – color assignments
  - $\text{dummycoloring}_a(v, c_1)$ – dummy action
  - $\text{vertex}_a(v)$ – vertex activation
- **Objective function:** $f(SS) = |\{\textbf{Do } \text{coloring}_a(v, c) \in SS\}|$

POSS: co-NP-Hardness (Theorem) – Part 2

Step 3: Integrity Constraints (encode graph structure)

- $\leftarrow \text{coloring}(X, C_1) \wedge \text{dummycoloring}(Y, C_2)$
  (Can't have both real and dummy coloring)
- $\leftarrow \text{edge}(X, Y) \wedge \text{coloring}(X, C) \wedge \text{coloring}(Y, C)$
  (Adjacent vertices can't share colors – **encodes edge constraints**)
- $\leftarrow \text{coloring}(X, c_1) \wedge \text{coloring}(X, c_2)$ (each vertex gets one color)

Step 4: Key Equivalence

G is 3-colorable $\Leftrightarrow \exists SS'$ with $f(SS') = |V| \Leftrightarrow$ empty SS is NOT Pareto-optimal

Step 5: Conclusion

Since 3-colorability is NP-complete, determining if SS is Pareto-optimal is co-NP-hard.

POSS: Corollary — co-NP-Completeness

Corollary: Under fixed program, constraints, and polynomial objectives, POSS membership is **co-NP-complete**.

Proof:

1. **Upper bound:** Proposition 1 shows membership in co-NP
2. **Lower bound:** Theorem 1 shows co-NP-hardness
3. **Therefore:** co-NP-complete

Practical Implication:

- Unless P = NP, no polynomial algorithm exists
- Motivates approximate/heuristic algorithms
- Our algorithms exploit problem structure for practical efficiency

POSS: Closure Complexity

Proposition: Under fixed program and constraints, Closure runs in **polynomial time**.

Proof Sketch:

1. Initialization: $O(|A|)$ where A is action set
2. Main loop iterations:
   - Each iteration adds at least one status atom
   - Maximum status atoms: $O(|A|)$
   - Therefore: $O(|A|)$ iterations
3. Per-iteration cost:
   - Rule application: $O(|P|)$ for fixed program P
   - Conflict checking: $O(|DC|)$ for fixed denials
4. Total: $O(|A| \cdot (|P| + |DC|)) =$ polynomial

POSS Algorithms: POSS-WAM (Weakly Anti-Monotonic)

Key Idea: Traverse lattice of status sets bottom-up (BFS)

Why this works for anti-monotonic functions:

- If $SS_1 \subseteq SS_2$, then $f(SS_1) \geq f(SS_2)$ (anti-monotonic)
- Smaller status sets have higher (better) objective values
- So start from smallest (LSS) and expand only if needed

Algorithm Flow:

1. Compute LSS = minimal status set satisfying rules
2. If LSS feasible and Pareto-optimal, return it
3. Otherwise, expand by adding one status atom at a time
4. At each level, check for feasible Pareto-optimal sets
5. First feasible set found is Pareto-optimal (due to anti-monotonicity)

Efficiency: Avoids enumerating all $2^{|A|}$ possible status sets

POSS Algorithms: POSS-SAM (Strongly Anti-Monotonic)

Key Difference from POSS-WAM:

- Strongly anti-monotonic: only **Do** atoms affect objective value
- $\{\alpha | \mathbf{Do}\alpha \in SS_1\} \subseteq \{\alpha | \mathbf{Do}\alpha \in SS_2\} \Rightarrow f(SS_1) \geq f(SS_2)$

Algorithm Optimization:

1. Separate status atoms into:
   - $SA\text{-}Do = \{\mathbf{Do}\alpha\}$ atoms
   - $SA\text{-}FPO = \{\mathbf{F}\alpha, \mathbf{P}\alpha, \mathbf{O}\alpha\}$ atoms
2. Initialize with **all** FPO combinations: $\{LSS \cup X | X \subseteq SA\text{-}FPO\}$
3. Expand by adding only **Do** atoms (not all status atoms)
4. This reduces search space significantly

Intuition: Since only **Do** atoms affect objectives, we can freely add FPO atoms without changing Pareto-optimality comparisons.

DEWS: No-Fly Zone Feature Computation

Data Source: godrone.nl (Dutch no-fly zone database)

Distance Computation: Haversine formula for GPS coordinates

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

where r = Earth's radius, ϕ = latitude, λ = longitude

Six NFZ Features Extracted:

1. `enter_noflyzone`: Binary flag (1 if trajectory entered any NFZ)
2. `perc_noflyzone`: Percentage of waypoints inside NFZ
3. `nf_d_min`: Minimum distance to nearest NFZ boundary
4. `nf_d_max`: Maximum distance to nearest NFZ boundary
5. `nf_d_mean`: Mean distance to nearest NFZ
6. `nf_d_std`: Standard deviation of NFZ distances

Surprising Finding: NFZ features less important than expected; asset values dominate.

DEWS: Late Fusion Formula

$$y(t) = \sum_{i=1}^{11} M_i(t) \cdot w_i$$

where:

- $M_i(t)$ = probability prediction from classifier i that trajectory t is threatening
- w_i = weight assigned to classifier i
- $\sum_{i=1}^{11} w_i = 1$ (weights normalized)

Weight Optimization:

- Grid search over weight combinations
- Objective: maximize overall F1-score on validation set
- Final weights emphasize classifiers with complementary errors

Why Late Fusion Works:

- Ensemble reduces variance and improves robustness
- Consistently outperforms all 11 individual classifiers

DEWS: Runtime Analysis

Two-Phase Runtime:

1. **Feature Extraction:** Slight increase with observation window
   - More waypoints = more computation
   - Still sub-second for all windows
2. **Prediction (Late Fusion):** Constant regardless of trajectory length
   - Fixed number of features (110)
   - Same 11 classifiers regardless of input size

Why 3 seconds total runtime is acceptable:

- DEWS is early warning, not real-time interception
- +7% F1 improvement over best single classifier
- Enables proactive defense planning

DEWS: Why F1 Drops at Intermediate Windows

Finding 2: Increasing observation window does NOT always improve performance

Observed Pattern:

- 5s: Recall = 0.789, Precision = 0.934
- 30-60s: Slight decline in both metrics
- 180s+: Performance improves again
- Peak at 360s (6 min)

Explanation: Trajectory Heterogeneity

- **At 5s:** Only fast-moving, clearly threatening trajectories captured → high precision
- **At 30-60s:** Mix of:
  - Fully-observed short trajectories (complete information)
  - Partially-observed long trajectories (incomplete information)
  - This heterogeneity creates classification noise
- **At 180s+:** Most trajectories fully characterized → cleaner signal

Dataset Statistics: Avg. trajectory duration 265-298 seconds

STATE: Why CLIP Encoder?

Answer: CLIP provides powerful *visual* feature extraction

Ablation Study Results:

Configuration	MDE (Threat)	MDE (Safe)
Full STATE (CLIP)	**1.27**	**1.62**
STATE w/o CLIP (histogram)	13.34	19.94
STATE w/o F (no geography)	8.64	6.25

Key Insight: Histogram encoding is WORSE than no geography!

Why CLIP works for our task:

- 9-channel tensor F (satellite, street map, NFZ, population, assets) is complex
- CLIP's pre-training on diverse image-text pairs $\rightarrow$ robust features
- Simple encoders cannot capture high-level spatial correlations
- CLIP generalizes to "atypical" visual inputs like our geographic features

STATE: Loss Components and Weights

Combined Generator Loss:

$$\mathcal{L}_G = \lambda_V \cdot \mathcal{L}_V + \lambda_T \cdot \mathcal{L}_T$$

Component Roles:

1. $\mathcal{L}_V$ (Trajectory Validity Loss):
   - Feedback from discriminator $\mathcal{M}^V$
   - Ensures generated trajectories look realistic
   - "Can this fool the discriminator?"
2. $\mathcal{L}_T$ (Threat Alignment Loss):
   - Feedback from pre-trained classifier $\mathcal{M}^T$
   - Ensures trajectories match target threat class
   - "Does this look threatening/safe as intended?"

Weight Rationale ($\lambda_V = 0.6$, $\lambda_T = 0.4$):

- $\lambda_V > \lambda_T$: Prioritize realism (unrealistic trajectories useless)
- Too high λ_T: Generator places disproportionate waypoints in NFZ
- Balance determined empirically

STATE: Why Keep $\mathcal{M}^T$ Static?

Design Choice: $\mathcal{M}^T$ is *pre-trained and fixed*, NOT adversarial

Evidence from Adversarial Training Analysis:

Beginning of Training:

- Uses threat label $\hat{\theta}$ effectively
- Clear separation in embedding space

End of Training:

- $\mathcal{M}^V$ can no longer separate threat classes
- Prioritizes features for real/fake distinction
- Threat semantics lost

Critical Insight:

"$\mathcal{M}^V$ prioritized features that do not depend on threat semantics to distinguish real from synthetic."

Solution: Separate threat-aware feedback via static $\mathcal{M}^T$

- $\mathcal{M}^T$ always knows what "threatening" looks like
- Decouples realism from threat alignment
- Both objectives jointly optimized by generator

Statistical Methods Used

All Key Results Are Statistically Significant

Method: Mann-Whitney U-test

- Non-parametric test (no normal distribution assumption)
- Compares two independent samples
- Appropriate for our experimental data

Bonferroni Correction:

- Adjusts for multiple hypothesis testing
- If testing n hypotheses at significance α:

$$\alpha_{\text{corrected}} = \frac{\alpha}{n}$$

- Prevents false positives from multiple comparisons

STATE: Statistical Significance Results

Table Caption: "*** indicates statistical significance (Bonferroni-corrected p-value < 0.001)"

Key Significant Results:

Comparison	Metric	Values	Significance
STATE vs LSTM	MDE (threat)	1.27 vs 5.25	$p < 0.001$***
STATE vs LSTM	MDE (safe)	1.62 vs 3.19	$p < 0.001$***
STATE vs VAE	MDE (threat)	1.27 vs 9.61	$p < 0.001$***
STATE vs all	SSIM	0.66 vs >0.85	$p < 0.001$***

Quote from Dissertation:

"All hypotheses tested in this chapter report Bonferroni-corrected p-values, obtained with Mann-Whitney U-test, to adjust for multiple hypothesis testing."

GUARDIAN: Mean Q-Values

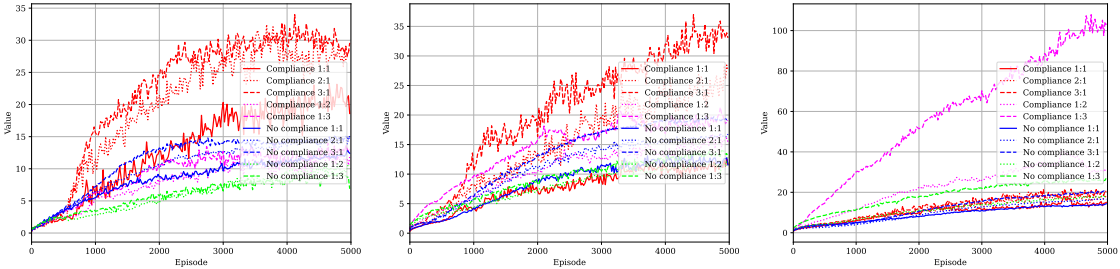


Figure: 64×64 grid: 16, 32, 64 BLUE drones.

GUARDIAN: Mean Q-Value Compliance Cost

B:R	16 BLUE 64×64	16 BLUE 128×128	32 BLUE 64×64	32 BLUE 128×128	64 BLUE 64×64	64 BLUE 128×128
			Mean Q-Values (higher = better)			
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: Compliance cost ratio for mean Q-values: > 1 means compliance *increases* learned value estimates.

Key Findings:

- Compliance usually increases mean Q-values, with gains up to $\sim 3.4\times$ in the best settings.
- Only 3 of 30 configurations show $CC_Q < 1$ (slightly lower Q-values under norms).
- Largest boosts occur for B:R = 1:2 or 1:3 with more BLUE drones and larger grids, suggesting norms help focus value learning in harder scenarios.

GUARDIAN: Action Entropy

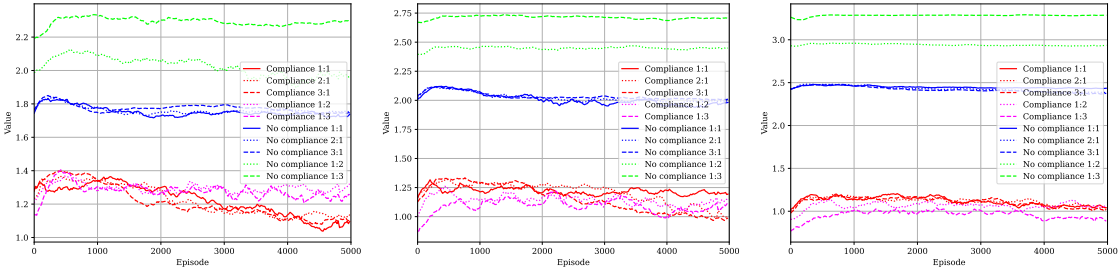


Figure: 64×64 grid: 16, 32, 64 BLUE drones.

GUARDIAN: Action Entropy Compliance Cost

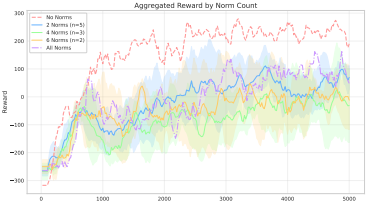
B:R	16 BLUE 64×64	16 BLUE 128×128	32 BLUE 64×64	32 BLUE 128×128	64 BLUE 64×64	64 BLUE 128×128
			Action Entropy (lower = better)			
1:1	0.698	0.748	0.607	0.629	0.457	0.501
2:1	0.694	0.759	0.596	0.619	0.460	0.494
3:1	0.683	0.748	0.567	0.623	0.459	0.500
1:2	0.640	0.634	0.467	0.502	0.362	0.381
1:3	0.557	0.571	0.402	0.462	0.292	0.327

Table: Compliance cost ratio for action entropy: < 1 means compliance *reduces* entropy (more decisive policies).

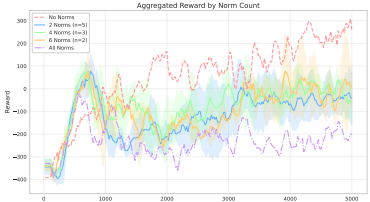
Key Findings:

- All configurations have $CC_H < 1$: norms consistently lower action entropy (policies less random / more focused).
- Strongest reductions (down to $\sim$0.29–0.33) occur for B:R = 1:3 with many BLUE drones, indicating sharper decisions when defenders are outnumbered.

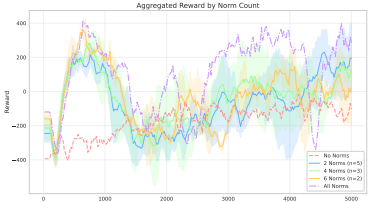
GUARDIAN: Norm Combinations



(a) 64×64 grid, 16v16 drones



(b) 64×64 grid, 32v32 drones



(c) 64×64 grid, 64v64 drones

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

Limitations: Dataset and Generalizability

DEWS & STATE Limitations:

- Single-city dataset (The Hague)
- 349 trajectories over 8 months
- May not generalize to:
  - War zones (Ukraine)
  - Different urban layouts
  - Different drone types

Mitigating Factors:

- Key predictor (asset values) doesn't depend on drone
- Tested with three distributions (LTP, MTP, HTP)
- STATE designed to generalize to unseen regions
- Expert evaluation validates generalization

GUARDIAN Limitations:

- 2D grid (no altitude)
- RED as non-learning baseline

Limitations: Incrementally Adding New Norms

Question: Can new norms be added without retraining?

Current Answer: No – retraining is required

Why Retraining is Necessary:

1. CAS computation happens at each timestep
2. Norms affect action masking during training
3. New norms change feasible action space
4. Policy must relearn optimal behavior in new space

Future Work:

- Transfer learning for norm updates
- Modular policy architectures
- Incremental norm incorporation

Limitations: Training Parameters

Question: Why 5,000 episodes? Why these parameters?

5,000 Episodes Justification:

- Standard RL practice for multi-agent environments
- Observed convergence in learning curves (typically by episode $\sim$4,000)
- Computational constraint: 630 hours with DRs vs 80 hours without

Other Fixed Parameters:

Parameter	Value	Rationale
Grid sizes	64×64, 128×128	Standard benchmarks
Battery	100 units	Reasonable operational constraint
View range	5 cells	Balance realism/tractability
Fire range	1 cell	Close-range engagement
Communication	80% success	Models realistic unreliability

Q&A: Why Not Use Speed in Trajectory Definition?

Question: Your trajectory definition has no speed. Isn't that a limitation?

Answer:

- **Acknowledged limitation** (mentioned in dissertation)
- **Current definition:**

$$\tau = \{w_j = (\text{lat}_j, \text{long}_j, h_j) | j = 1, \ldots, M_\tau\}$$

- **Why acceptable for our work:**
  - DEWS extracts speed as derived features from timestamps
  - STATE focuses on spatial patterns, not temporal dynamics
  - Altitude variations captured in h_j
- **Future work:**
  - Extend to include timestamps: $(\text{lat}_j, \text{long}_j, h_j, t_j)$
  - Model velocity and acceleration patterns
  - Enable temporal trajectory generation

Q&A: Why Disjoint Train/Test Regions?

Question: "A disjoint from C" – what does this mean?

Context: STATE training vs. testing regions

Answer:

- Training regions $\mathcal{A}$ and testing regions $\mathcal{C}$ are *disjoint*
- No overlap between where we train and where we test
- This is *critical* for validating generalization

Why this matters:

- If $\mathcal{A} \cap \mathcal{C} \neq \emptyset$: Model might memorize specific regions
- Disjoint regions ensure model learns *general* patterns
- Validates STATE's use case: generating for unmonitored areas

From dissertation:

> *"This mirrors our use case for STATE: synthesizing plausible drone trajectories for regions where no flight data is available."*

Q&A: Dual-Use Concerns

Question: Could STATE be misused to plan attacks?

Answer:

- **Acknowledged concern** (mentioned in dissertation limitations)
- **Mitigating factors:**
  - STATE designed for defense testing, not attack planning
  - Requires asset value annotations (controlled by authorities)
  - Generating trajectories $\neq$ operational capability
  - Similar dual-use exists in all security research
- **Responsible disclosure:**
  - Developed with Dutch Police input
  - They understand and accept the research
  - Benefits to defense outweigh risks
- **Privacy protections:**
  - Features can exclude identifying information
  - Compatible with privacy-preserving approaches

Q&A: Why Binary Threat Classification?

Question: Why only safe vs. threatening? What about medium threats?

Answer:

- **DEWS handles multiple levels:**
  - LTP: Low-threat prediction (score < 4)
  - MTP: Medium-threat prediction (score $\in [4, 8)$)
  - HTP: High-threat prediction (score ≥ 8)
- **STATE uses binary for simplicity:**
  - Proof of concept for conditioned generation
  - Binary distinction is most operationally relevant
  - Multi-class generation is straightforward extension
- **Operational reality:**
  - Security responses are often binary (intervene or not)
  - Medium threats can be handled by adjusting threshold
  - DEWS provides continuous threat scores for nuance

Q&A: Why Not Reward Shaping?

Question: Why use hard constraint enforcement instead of reward shaping for compliance?

Answer:

- **Reward shaping:** Violations incur penalties but remain possible
  - Agent may learn to "accept" penalty for high-reward violations
  - No formal guarantee of compliance
  - Sensitive to penalty magnitude tuning
- **Hard constraints (our approach):**
  - Infeasible FSSs *never* explored during training or execution
  - Formal guarantee: all policies are compliant by construction
  - Reduces search space (can improve learning)

Our experiments show: Hard constraints can actually *improve* performance at scale.

Q&A: Why RL Instead of Pareto Optimization?

Question: POSS uses Pareto optimization. Why doesn't GUARDIAN?

Pareto Optimization (POSS):

- Static: One decision point
- Objectives fixed a priori
- No feedback from environment
- Optimal for *this moment*

Use case: Single resource allocation

RL (GUARDIAN):

- Sequential: Many decisions
- Learn from outcomes
- Environment feedback (rewards)
- Optimal over *trajectory*

Use case: Multi-step game

Key: Defense is inherently sequential. Today's action affects tomorrow's state. RL captures this; static Pareto doesn't.

Q&A: What GUARDIAN Keeps from POSS

GUARDIAN uses POSS for constraint enforcement, not optimization.

Component	From POSS?	Purpose in GUARDIAN
Deontic operators (**P**, **O**, **F**, **Do**)	✓	Specify legal constraints
Status set definition	✓	Structure for action bundles
Feasibility conditions	✓	Check legal compliance
Closurealgorithm	✓	Compute minimal legal set
FSS enumeration	✓	Find all legal options
Multi-objective functions	×	Replaced by RL reward
Pareto dominance check	×	Replaced by Q-learning

Summary: POSS provides the "constraint engine"; RL provides the "policy engine."