Safety Evaluation of Neural Network Control for Drone Delivery Systems

Calum Arnott*[1,2], Julia López Gómez*[1,2], and Ruchira Ray*[1,2]

[1] University of Edinburgh, UK
{C.R.Arnott,J.Lopez-Gomez,R.Ray-3}@sms.ed.ac.uk
[2] Heriot-Watt University, UK
{cra4000,jl4065,rr4033}@hw.ac.uk

Abstract. Neural network controllers are increasingly used in safety-critical cyber-physical systems due to their ability to approximate complex non-linear control policies. However, providing guarantees about their behaviour remains challenging, particularly when such controllers are deployed in closed loop with non-linear dynamics. This paper presents a case study on the robustness and verification of a neural network controller for a quadrotor drone performing winch-based payload delivery with a tethered load. An expert PID controller is first designed to stabilise the coupled drone-payload system in desired setpoints and is used to generate training data. A NN controller is then trained to approximate the expert policy, and adversarial training and property-driven training are incorporated to improve robustness and bias the controller toward safe behaviour and property compliance. The learned controllers are evaluated using three complementary verification techniques. First, controller-level formal property verification is performed using the Vehicle specification language and the Marabou solver to assess properties such as bounded actuation or correct control direction. Second, finite-time reachability analysis of the closed-loop system is conducted using CORA, revealing scalability limitations due to non-linear dynamics and dimensionality. Third, empirical robustness is evaluated under adversarial input perturbations, complemented by explainability analysis using LIME and Integrated Gradients to assess feature sensitivity and stability. This case study highlights the interaction between training strategies and verifiability, and showcases current challenges in verifying NN controllers for aerial robotics applications.

Keywords: Neural Network Control · Uncrewed Aerial Vehicle · Verification of Cyber-Physical Systems · Machine Learning.

1 Introduction

We introduce a novel verification case study for an aerial delivery cyber-physical system: a planar quadrotor with an underslung payload connected via a tether.

* Equal contribution

The benchmark preserves key non-linear couplings between vehicle motion, payload swing, and tether dynamics while remaining sufficiently structured to integrate with current neural network verification tools. Using this benchmark, we develop an expert PID controller and train a neural network controller by behavioural cloning, providing a learned control policy suitable for both empirical evaluation and formal analysis.

To improve robustness and encourage safe behaviour, we investigate robustness-based training, including adversarial training under bounded input perturbations, and property-driven training (PDT), in which controller-level specifications are translated into differentiable penalty terms and integrated into the learning objective. We analyse how these training strategies affect both empirical robustness and verifiability. To support interpretability and diagnostic insight, we apply explainability analysis using LIME [26] and Integrated Gradients [29] to compare feature importance and stability between baseline and robustly trained controllers.

Finally, we evaluate the learned controllers using three complementary assurance techniques at different levels of abstraction: (i) controller-level formal property verification of the neural network in isolation using the VEHICLE specification language [6] with the MARABOU solver [30]; (ii) system-level finite-horizon closed-loop reachability analysis using CORA [1]; and (iii) empirical robustness evaluation via adversarial perturbation testing. All code for simulation, training, and verification is released as open source[3].

1.1 Drone Delivery

The use of Uncrewed Aerial Vehicles (UAVs)–more commonly known as drones–for the delivery of packages has become increasingly widespread in both rural and urban environments. Drone delivery has been used to improve access to healthcare and ensure time-critical delivery of medical goods, such as vaccines [11,7,22] and blood [21,16,9,3]. In urban environments, drones are increasingly being developed for last-mile delivery of food and consumer goods [4,19]. Operation of drones, particularly in urban environments, requires adherence to strict safety and reliability requirements administered by local flight aviation authorities.

Several mechanisms have been utilised for delivering packages using drones, including parachute-based delivery [9,12,13] and servo-actuated mechanisms [18,14]. Whilst these approaches offer a straightforward method for dropping the payload ('package'), neither offers precise control over the payload descent dynamics. Increasingly, winch based mechanisms are explored in which the payload is suspended beneath the drone on a tether and lowered to the ground using a motorised winch [24,5,23]. This configuration has several practical advantages, including the ability to deliver a payload without requiring a human to approach the drone itself, thereby reducing the risk of collision with hazardous rotating components such as propellers. Nevertheless, winched delivery introduces safety

[3] `https://github.com/calumarnott/f21zk.git`

concerns to address, including payload swing, slack or excessive tension in the tether, winch overload, and collisions with the surrounding environment.

The resulting drone-payload system shows coupled multi-body dynamics along with non-linear effects. Ensuring safe operation therefore requires careful control design and rigorous analysis.

1.2 Neural Network Control of Cyber-Physical Systems

Neural Networks (NNs) have been utilised for control in cyber-physical systems [20], including for UAV applications [2,31]. NNs are appealing due to their ability to learn complex non-linear control mappings, to cope with uncertainty, and to perform well when accurate modelling of system dynamics is difficult [10]. These properties are particularly relevant in aerial robotics, where system dynamics include coupled translational and rotational motion, non-linear actuator behaviour, and environmental disturbances such as wind. However, the adoption of NN controllers in safety-critical applications remains limited by the lack of formal guarantees on their behaviour. Verification of NNs as controllers is a burgeoning research field [28,27,8], and formal verification techniques are desirable for assuring safety in systems that operate in close proximity to people and the environment, such as delivery drones.

From a verification perspective, the drone delivery system with an underslung payload presented has several distinguishing challenges from existing cyber-physical system (CPS) benchmarks. Firstly, the NN controller is based on a regression network, which produces continuous-valued control outputs rather than discrete actions or class labels. Moreover, the closed-loop dynamics of the combined drone–payload system are non-linear and relatively high-dimensional, making reachability analysis computationally challenging. Additionally, safety is not naturally expressed as simple avoidance of unsafe regions in the state space; rather, desired behaviour is more naturally characterised as maintaining bounded deviation from a target trajectory or operating regime. Though prior benchmarks have addressed the verification of neural network controllers for a quadcopter drone [17], the inclusion of an underslung payload increases the complexity of the system as the controller must simultaneously regulate the motion of the drone and the payload, which are highly coupled dynamically.

This places the drone-payload system at the boundary of what existing neural network verification and reachability tools can handle in practice.

2 Modelling

We model a quadcopter drone transporting a payload via a tethered winch mechanism. To improve tractability for verification, the dynamics are simplified with the following assumptions: planar motion in the x–z plane; a rigid, symmetric quadcopter with mass concentrated at its centre of mass; a massless, inextensible tether that remains taut; pure axial thrust from the rotors; and no aerodynamic drag or external disturbances. The tether tension is resolved via a quasi-static

iteration at each timestep, assumed accurate for sufficiently small integration steps. These assumptions preserve the dominant non-linear couplings between the drone and payload while yielding a model suitable for formal analysis.

2.1 State, Controls, and Parameters

The system state captures the planar motion of the quadcopter and the dynamics of the tethered payload:

$$\boldsymbol{s} = \left[x_q \; z_q \; \theta \; \dot{x}_q \; \dot{z}_q \; \dot{\theta} \; \ell \; \phi \; \dot{\ell} \; \dot{\phi}\right]^\top .$$

The control input consists of the individual rotor thrusts and the commanded tether feed rate:

$$\boldsymbol{u} = \left[T_1 \; T_2 \; T_3 \; T_4 \; u_\ell\right]^\top .$$

Here, x_q and z_q denote the quadcopter centre-of-mass position, θ is the pitch angle, ℓ is the tether length, and ϕ is the payload swing angle measured from the vertical. Dots denote time derivatives.

The system's physical parameters are the quadcopter mass m_q, payload mass m_p, pitch-axis inertia I_q, gravitational acceleration g, and rotor arm length d.

2.2 Equations of Motion

The translational dynamics of the quadcopter are given by

$$m_q \begin{bmatrix} \ddot{x}_q \\ \ddot{z}_q \end{bmatrix} = \boldsymbol{F}_{\text{thrust}} + Q\,\boldsymbol{u}(\phi) - m_q g \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{1}$$

where $\boldsymbol{F}_{\text{thrust}}$ is the total thrust force and Q is the tether tension.

The rotational dynamics about the pitch axis are

$$I_q\,\ddot{\theta} = \tau, \tag{2}$$

with τ denoting the applied pitch torque.

The payload dynamics satisfy

$$m_p\,\ddot{\boldsymbol{p}}_p = -m_p g \begin{bmatrix} 0 \\ 1 \end{bmatrix} - Q\,\boldsymbol{u}(\phi), \tag{3}$$

where $\boldsymbol{p}_p$ is the payload position.

Projecting the payload dynamics onto the tether frame yields the swing equation

$$\ell\,\ddot{\phi} + 2\,\dot{\ell}\,\dot{\phi} = -g\,\sin\phi + \begin{bmatrix} \ddot{x}_q \\ \ddot{z}_q \end{bmatrix} \cdot \boldsymbol{t}(\phi), \tag{4}$$

and the radial equation

$$m_p(\ddot{\ell} - \ell\dot{\phi}^2) = m_p g\cos\phi - Q + m_p \begin{bmatrix} \ddot{x}_q \\ \ddot{z}_q \end{bmatrix} \cdot \boldsymbol{u}(\phi). \tag{5}$$

Solving for the tether tension gives

$$Q = m_p \left[g \cos\phi + \begin{bmatrix} \ddot{x}_q \\ \ddot{z}_q \end{bmatrix} \cdot \boldsymbol{u}(\phi) - (\ddot{\ell} - \ell\dot{\phi}^2) \right]. \tag{6}$$

The tether feed rate is controlled directly by the winch input,

$$\dot{\ell} = u_\ell. \tag{7}$$

All geometric definitions and intermediate derivations are provided in Appendix 7.3.

3 Implementation

This section describes the expert controller and simulation framework used to generate training data, followed by the learning pipeline used to train a regression neural network controller via behavioural cloning.

3.1 Expert PID Controller

Each controlled variable x_i is governed by a PID law of the form

$$u_i(t) = K_{p,i} e_i(t) + K_{i,i} \int_0^t e_i(\tau)\, d\tau + K_{d,i} \frac{de_i(t)}{dt}, \tag{8}$$

where $e_i(t) = x_{i,\mathrm{ref}}(t) - x_i(t)$ denotes the tracking error.

Independent PID controllers are implemented for the following objectives:

$$\begin{aligned} &u_x : \text{horizontal quadcopter position } (x_q), \\ &u_z : \text{vertical quadcopter position } (z_q), \\ &u_\phi : \text{payload swing damping } (\phi \to 0), \\ &u_\theta : \text{quadcopter pitch stabilisation } (\theta), \\ &u_\ell : \text{tether feed control } (\ell_{\mathrm{ref}}(t)). \end{aligned}$$

A cascaded control structure is used, with outer-loop controllers regulating position and payload motion, and inner-loop controllers stabilising vehicle attitude and velocity. Additional details of the implementation are included in Appendix 7.4. For simplicity, and to reduce the dimensionality of the learning problem, the neural network controller is trained assuming a fixed tether length; consequently, the tether feed controller is inactive during data generation.

The PID controller is not intended to be optimal, but rather to provide a reliable and interpretable expert suitable for supervised learning. As a result, the behaviour and safety properties of the learned neural network controller are inherently constrained by the coverage and quality of the expert demonstrations.

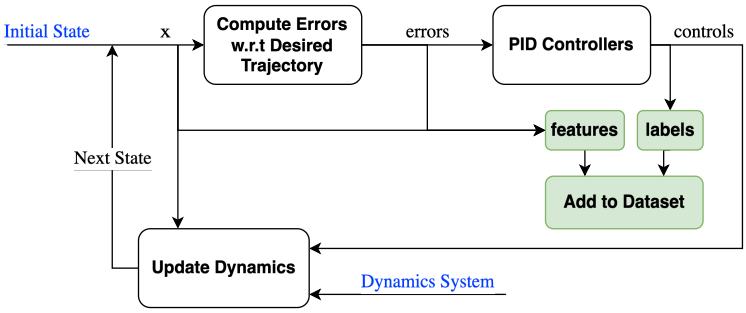


Fig. 1: Dataset generation pipeline using expert PID controller in simulation. The numerical simulation framework used with the expert PID controller to update the system dynamics is detailed in Appendix 7.1.

3.2 Dataset Generation

The expert PID-controlled simulation is used to generate training data for the NN controller. Trajectories are generated by sampling a range of initial conditions to reach a reference setpoint, and logging the resulting state, error, and control tuples at each simulation step. The data generation pipeline is outlined in Fig. 1. At each time step t, the input to the learning model –the dataset features– consists of the system state $\boldsymbol{s}_t$ together with a vector of tracking errors $\boldsymbol{e}_t$, defined with respect to the desired reference trajectory or setpoint:

$$\boldsymbol{e}_t = \boldsymbol{s}_{\text{ref}}(t) - \boldsymbol{s}_t.$$

The resulting dataset is given by

$$\mathcal{D} = \{(\boldsymbol{s}_t, \boldsymbol{e}_t), \boldsymbol{u}_t\}_{t=0}^{T},$$

where $\boldsymbol{u} = \pi_{\text{PID}}(\boldsymbol{s})$ denotes the control action generated by the PID controller.

3.3 Neural Network Controller

The learned controller is implemented as a feed-forward neural network that approximates the expert PID policy via supervised behavioural cloning. The network maps the current system state and tracking errors to continuous-valued control outputs, and is evaluated in closed loop using the nonlinear system dynamics described in Section 2.

Architecture The controller is implemented as a fully connected multi-layer perceptron (MLP). The network takes as input the concatenation of the system state $\boldsymbol{s}_t$ and the corresponding tracking error vector $\boldsymbol{e}_t$, and outputs continuous-valued control commands:

$$f_\theta : (\boldsymbol{s}_t, \boldsymbol{e}_t) \mapsto \boldsymbol{u}_t.$$

The architecture consists of three 32–node hidden layers, each followed by a ReLU activation. Dropout can optionally be applied during training to improve generalisation. The output layer is linear, as the control task is formulated as a regression problem.

Training Procedure Training is done via supervised learning on the generated dataset. The network parameters θ are optimised to minimise a mean squared error (MSE) loss between the predicted control outputs and the expert data:

$$\mathcal{L}_{\text{MSE}}(\theta) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{e},\boldsymbol{u})\sim\mathcal{D}}\left[\|f_\theta(\boldsymbol{s},\boldsymbol{e}) - \boldsymbol{u}\|_2^2\right].$$

The dataset is split into training, validation, and test subsets (80/10/10% respectively). We train for 100 epochs using a standard gradient-based optimiser, with early stopping based on validation loss to mitigate overfitting. Model performance is evaluated using both mean squared error (MSE) and mean absolute error (MAE) on held-out test data.

Closed-Loop Deployment After training, the neural network controller replaces the expert PID controller in the closed-loop simulation. At each control step, the current system state and tracking error are passed to the network to generate control commands, which are then applied to the system dynamics.

Although the network is trained to imitate a stabilising expert policy, no formal guarantees of stability or safety are provided by the learning process alone. In particular, the learned controller may exhibit unstable behaviour for states outside the training distribution or under perturbations, motivating the robustness analysis and formal verification presented in later sections.

Adversarial and Property-Driven Training (PDT) Beyond standard supervised training, additional training objectives are introduced to improve robustness and enforce safety-relevant properties in the learned controller. Two complementary strategies are considered: adversarial training and property-driven training. The integration of these strategies into the full closed-loop NN controller system is shown in Fig. 2.

The effectiveness of these strategies is evaluated through different verification techniques in Section 4.

3.4 Adversarial Training and Adversarial Example Generation

This section describes the adversarial training procedure used to improve the robustness of the learned controller, as well as the method for generating adversarial examples during verification and evaluation.

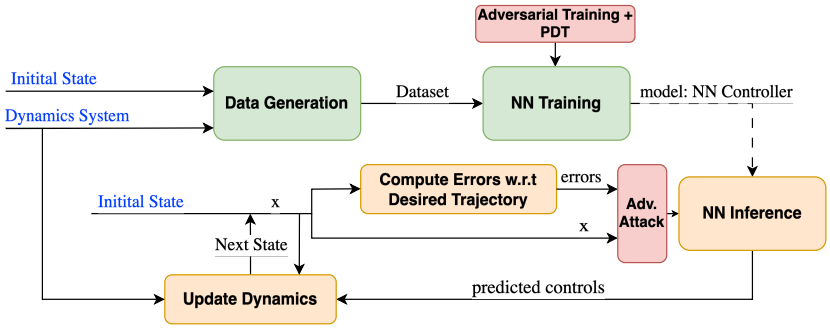


Fig. 2: Neural network controller pipeline with adversarial and property-driven training. Adversarial training and PDT are applied during model training. During inference, the trained network is deployed in closed loop with the system dynamics, and adversarial examples are generated by perturbing the input state within a bounded set. This allows for controllers, trained with or without adversarial objectives or PDT, to be evaluated for robustness to noise.

Threat model and attack objective Let the controller be a regression model $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ mapping an input feature/state vector x to a control output $\hat{y} = f_\theta(x)$. We consider worst-case bounded perturbations δ applied to the input,

$$x_{\text{adv}} = x + \delta, \qquad \|\delta\|_\infty \le \varepsilon \quad \text{or} \quad \|\delta\|_2 \le \varepsilon, \tag{9}$$

and generate adversarial examples by maximising the MSE loss:

$$\max_{\|\delta\| \le \varepsilon} \mathcal{L}\big(f_\theta(x+\delta), y\big), \qquad \mathcal{L}(\hat{y}, y) = \|\hat{y} - y\|_2^2. \tag{10}$$

All reported attacks are *white-box*: they use gradients of $\mathcal{L}$ with respect to the perturbation (PGD) under the current model parameters.

Attacks considered We evaluate two standard white-box attacks adapted to regression, using MSE as the attack loss $\mathcal{L}(f_\theta(x), y) = \|f_\theta(x) - y\|_2^2$, where x is the clean input, y the target, and ∇_x denotes the gradient with respect to the input.

FGSM (single-step). FGSM takes one gradient-ascent step to increase $\mathcal{L}$. For an ℓ_∞ constraint,

$$x_{\text{adv}} = x + \varepsilon \cdot \text{sign}\big(\nabla_x \mathcal{L}\big(f_\theta(x), y\big)\big)\,, \tag{11}$$

where ε is the perturbation budget and $\text{sign}(\cdot)$ applies the gradient direction per feature. For ℓ_2, FGSM uses the normalised gradient direction.

PGD (iterative). PGD iteratively updates a perturbation δ_t for T steps and projects back to the ε-ball:

$$\delta_{t+1} = \Pi_{\|\delta\| \le \varepsilon}\Big(\delta_t + \alpha \cdot g_t\Big), \qquad x_{\text{adv}} = x + \delta_T, \tag{12}$$

where α is the step size and $\Pi_{\|\delta\|\le\varepsilon}$ enforces the norm constraint (clipping for ℓ_∞, rescaling for ℓ_2). The ascent direction is

$$g_t = \begin{cases} \operatorname{sign}\big(\nabla_\delta \mathcal{L}\big(f_\theta(x+\delta_t), y\big)\big), & \ell_\infty, \\ \dfrac{\nabla_\delta \mathcal{L}\big(f_\theta(x+\delta_t), y\big)}{\big\|\nabla_\delta \mathcal{L}\big(f_\theta(x+\delta_t), y\big)\big\|_2}, & \ell_2, \end{cases} \tag{13}$$

i.e., sign-gradient steps under ℓ_∞ and unit-gradient steps under ℓ_2.

We use *random start* by sampling δ_0 within the ε-ball to strengthen the attack.

Implementation details and evaluation protocol Adversarial inputs are generated via a unified attack factory (`make_attack_fn`) and applied *per mini-batch* during training/evaluation. Concretely, for each batch (X, y) we compute $X_{\text{adv}} = \texttt{attack_fn}(f_\theta, X, y)$ and then evaluate the model on X_{adv} to obtain adversarial MSE/MAE. Clean evaluation is performed by setting `attack_fn=None`.

For qualitative inspection, we also generate a single adversarial sample from the test set and plot the clean vs. adversarial feature vector (Figure 4a). This highlights how PGD produces structured, loss-increasing perturbations across features rather than unstructured random noise.

Adversarial training parameters. Our adversarial training configuration uses **PGD** with ℓ_∞ perturbations, perturbation budget $\varepsilon = 0.05$, and $T = 40$ update steps (with default $\alpha = \varepsilon/T$). We use `mode=mix` with $\lambda = 0.5$ (i.e., every sample has a 0.5 probability of being attacked when adversarial training is enabled).

When enabled, adversarial training forces the controller to fit not only the nominal data distribution, but also the worst-case perturbations within the chosen threat model. Empirically, this typically reduces the growth of adversarial error as ε increases (at the cost of additional computation, since generating X_{adv} requires multiple forward/backward passes per batch). The robustness curves in Figure 4b illustrate this effect by comparing a normally trained model and an adversarially trained model under PGD evaluation.

4 System Verification and Robustness Analysis

This section evaluates the safety and robustness of the learned neural network controller, discussing the implementation and analysis of three complementary verification methods: (i) formal controller-level verification in isolation, (ii) system-level reachability analysis of the closed-loop dynamics, and (iii) empirical robustness analysis of the neural network under bounded input perturbations.

Together, these methods provide insight into which safety-relevant properties can be formally guaranteed, which can only be assessed empirically, and where current verification tools encounter scalability limitations.

4.1 Controller-Level Property Verification (VEHICLE + MARABOU)

This subsection focuses on formal verification of the neural network controller in isolation, independent of the underlying system dynamics. The goal is to assess whether the learned controller satisfies high-level, safety-relevant properties over bounded input domains. Properties are specified using the VEHICLE specification language and verified using the MARABOU neural network verifier.

Verification Scope and Assumptions We consider the neural network controller as a static function $f_\theta : (\boldsymbol{s}, \boldsymbol{e}) \mapsto \boldsymbol{u}$, mapping system states and tracking errors to control outputs. Verification is performed over bounded regions of the input space corresponding to physically valid states. This controller-level analysis provides formal guarantees about instantaneous control outputs, which are necessary but not sufficient for overall system safety.

Safety and Behavioural Properties Based on physical actuator limits and desired control behaviour, we define the following properties.

Property P1: Global Saturation Safety. For all valid inputs, each control output must remain within the physical actuator limits, ensuring that the controller never commands infeasible actuator values:

$$\forall (\boldsymbol{s}, \boldsymbol{e}) \in \mathcal{X}, \quad u_i^{\min} \leq f_\theta(\boldsymbol{s}, \boldsymbol{e})_i \leq u_i^{\max}, \quad \forall i.$$

Property P1.2: Saturation Safety with Margin. A relaxed variant of Property P1 introduces a margin factor $m > 0$, allowing small deviations over actuator limits:

$$\forall (\boldsymbol{s}, \boldsymbol{e}) \in \mathcal{X}, \quad u_i^{\min} - m \leq f_\theta(\boldsymbol{s}, \boldsymbol{e})_i \leq u_i^{\max} + m, \quad \forall i.$$

Property P2: Correct Control Direction at Extremes When selected state variables (e.g. position or payload swing) are far from their reference values, the controller must apply corrective actions in the appropriate direction. For example, for pendulum swing angle error e_ϕ:

$$e_\phi > \epsilon \;\Rightarrow\; f_\theta(\boldsymbol{s}, \boldsymbol{e})_{u_\phi} < 0, \quad e_\phi < -\epsilon \;\Rightarrow\; f_\theta(\boldsymbol{s}, \boldsymbol{e})_{u_\phi} > 0,$$

where $\epsilon > 0$ is a threshold defining "far" from the reference. Similar conditions apply for other critical state variables. This property encodes basic physical intuition about stabilising behaviour.

Property P3: Quiet Control Near Equilibrium Near equilibrium, where tracking errors and velocities are close to zero, the controller should apply minimal control:

$$\|\boldsymbol{e}\|_\infty \leq \varepsilon \;\Rightarrow\; \|f_\theta(\boldsymbol{s}, \boldsymbol{e})\|_\infty \leq u^{\text{quiet}},$$

for small thresholds $\varepsilon, u^{\text{quiet}} > 0$. This property discourages oscillatory or unnecessarily aggressive control near the desired operating point.

Formal Verification with Vehicle and Marabou Each property is encoded in the Vehicle specification language using bounded quantification over the input space. The resulting constraints are passed to the Marabou verifier, which symbolically analyses the neural network to determine whether the properties hold for all inputs in the specified domain. We provide a full Vehicle specification of the above properties in Appendix 7.5.

Verification outcomes are reported as either *verified/SAT* (property holds for all inputs in the specified domain) or *violated/UNSAT* (a counterexample is found). Due to network size and input dimensionality, not all properties are verifiable for all models, highlighting scalability limitations of current neural network verification tools.

Property-Driven Training To improve satisfaction of the above properties, selected specifications are incorporated into the training process via property-driven training (PDT). Each property is transformed into a differentiable penalty term that augments the standard imitation loss. We showcase an example.

Let $\mathcal{L}_{\text{MSE}}$ denote the regression loss used in our system. The total training objective becomes:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda_1 \mathcal{L}_{\text{sat}} + \lambda_3 \mathcal{L}_{\text{quiet}},$$

where λ_i are weighting coefficients.

Saturation Penalty. Violations of actuator limits are penalised using ReLU:

$$\mathcal{L}_{\text{sat}} = \sum_i \max(0, f_\theta(\boldsymbol{s}, \boldsymbol{e})_i - u_i^{\max})^2 + \max(0, u_i^{\min} - f_\theta(\boldsymbol{s}, \boldsymbol{e})_i)^2.$$

Quiet Control Penalty. Near-equilibrium behaviour is encouraged by penalising control effort when errors are small:

$$\mathcal{L}_{\text{quiet}} = \mathbb{1}_{\|\boldsymbol{e}\|_\infty \leq \varepsilon} \, \|f_\theta(\boldsymbol{s}, \boldsymbol{e})\|_2^2.$$

These penalties act as soft constraints during training, biasing the learned controller toward satisfying the specified properties.

Verification Results and Limitations Due to the size of the training dataset, the dimensionality of the input space, and limited computing resources, full controller-level verification of our tethered drone NN controller was not tractable using Vehicle and Marabou. To validate the effectiveness of the proposed property-driven training approach, we evaluated it on a simpler pendulum control benchmark commonly used in neural network verification studies.

Results indicate that property-driven training improves satisfaction of the specified safety properties. In particular, after applying PDT, properties 1.2 and 3 were successfully verified, compared to only property 3 being verified in the baseline model. Furthermore, the number of counterexamples found for violated properties in a sample dataset decreased significantly, demonstrating enhanced controller behaviour in critical scenarios.

While these findings show the effectiveness of PDT in enhancing verifiability, they also highlight the challenges of scaling neural network verification to higher-dimensional, real-world control problems. This motivates future research into more efficient verification methods tailored to complex control systems.

4.2 Closed-Loop Reachability Analysis (CORA)

ß This subsection analyses the closed-loop behaviour of the drone-payload system using finite time-horizon reachability analysis. The CORA tool is used, implemented in MATLAB, to compute over-approximations of the reachable state sets for the non-linear system dynamics. The analysis is performed to assess whether all trajectories originating from a bounded initial set satisfy specified safety and goal conditions over a finite horizon.

Verification Scope and Assumptions The verification task is formulated by defining (i) an initial set, representing uncertainty in the initial conditions; (ii) a safe set, describing admissible regions of the state space; and (iii) an unsafe set, which must be avoided for the duration of the reachability horizon.

While the safe behaviour of aerial delivery systems may be more naturally expressed as bounded deviation from a reference trajectory, as in the gliding drone case study of Kessler et al. [15], our system involves a higher-dimensional state space and strongly coupled non-linear dynamics. As a result, we restrict the specification to state-space safety and avoidance properties that are tractable within CORA, and defer trajectory-based specifications to future work.

State-Space Dynamics The continuous-time dynamics derived in Section 2 are reformulated in state-space form for use with CORA's non-linear reachability solvers. The system is expressed as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}),$$

where $\boldsymbol{x}$ comprises the drone and payload states, and $\boldsymbol{u}$ represents either bounded control inputs (open-loop analysis) or feedback control laws (closed-loop analysis).

Reachability Specification Reachability is evaluated over a finite time horizon of $T_f = 4$ s using CORA's non-linear reachability analysis. The specification requires all trajectories starting from a bounded initial set to avoid unsafe states for the entire horizon and satisfy a goal (safe) condition during the final second. Appendix 7.6 provides a representative code snippet showing how initial sets, safety and avoidance properties, and time-bounded reachability specifications are encoded and verified in CORA for the closed-loop system.

Initial Set The nominal initial condition corresponds to a hovering quadcopter at $z_q = 5$ m with zero velocities, zero payload swing, and a tether length of $\ell = 1$ m. Bounded uncertainty is modelled by an interval of radius 0.01 applied uniformly to all 13 state dimensions (10 plant states and 3 PID integral states), yielding the initial set $\mathcal{X}_0$.

Goal (Safe) Set The goal set constrains the quadcopter centre-of-mass altitude to remain within a prescribed flight corridor during the final second of the horizon. Specifically,

$$z_q \in [2,\ 10],$$

while the lateral position x_q and all remaining state variables are left unconstrained. This specification reflects a simplified altitude safety envelope consistent with operational flight constraints.

Unsafe Set The unsafe set represents a near-ground crash region. Entry into this set is defined by

$$z_q \in [0,\ 1],$$

with all other state dimensions unconstrained. The size of this buffer accounts for the presence of the tethered payload, ensuring that potential payload-ground contact is captured even when the quadcopter itself remains airborne.

Control Inputs Closed-loop reachability is performed with no external inputs, as all control actions are generated internally by the embedded PID controller. Accordingly, the input set is defined as $\mathcal{U} = \{0\}$.

Reachability Results We perform closed-loop reachability analysis of the drone–payload system under the expert PID controller using CORA. Since CORA does not provide a native class for PID control, the control law is embedded directly into the system dynamics. In particular, the integral terms of the PID controllers are modelled as additional state variables with their own differential equations, increasing the system dimension from 10 plant states to 13 states. This explicit embedding of a cascaded PID controller into CORA constitutes a non-trivial extension of the standard modelling workflow and represents a novel aspect of this case study.

Fig. 3 shows the reachable sets of the closed-loop system projected onto the quadcopter position subspace. For the given initial set and specification, the safety properties are verified for a short initial time interval. In practice, reachability propagation can be carried out successfully up to approximately 0.39 s, after which set explosion prevents further computation, thereby limiting the achievable time horizon for closed-loop verification.

Discussion and Limitations Closed-loop reachability analysis with a neural network controller has not yet been implemented in CORA for this system. While

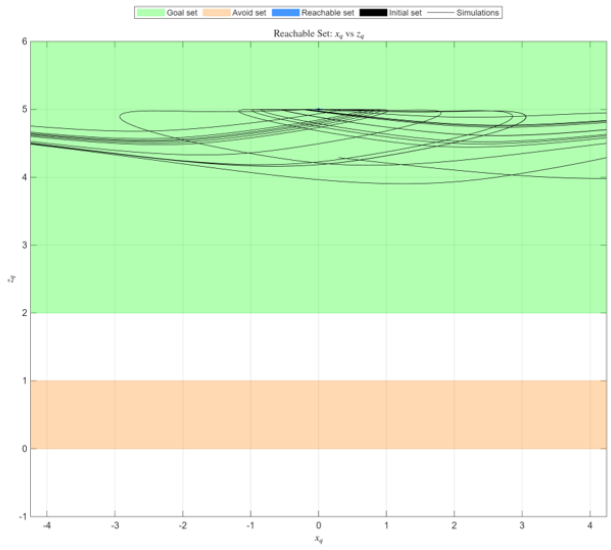
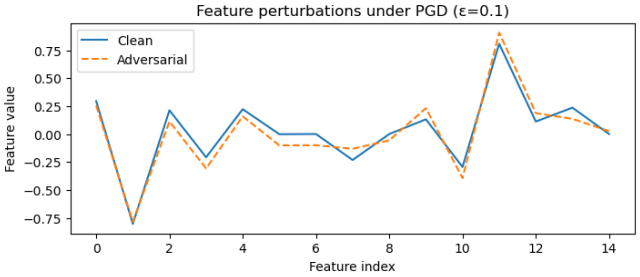


Fig. 3: Closed-loop reachability with PID control.

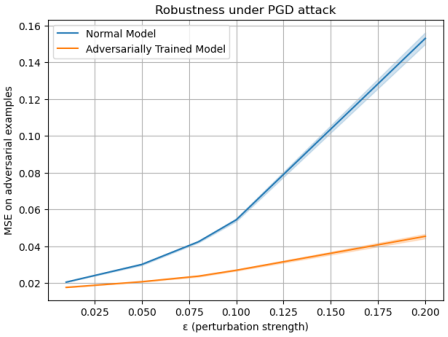
conceptually straightforward—replacing the PID control law with a neural network evaluation—the primary focus of this work was to characterise the onset of set explosion for the coupled drone–payload dynamics before introducing additional non-linearities from the neural network. The PID-based analysis already demonstrates severe scalability limitations, suggesting that further model simplifications or alternative reachability representations would be required for neural network closed-loop verification. These preliminary results in CORA highlight the challenges of system-level verification of closed-loop control with complex multi-body dynamics, and motivate the complementary use of controller-level verification and robustness analysis also presented in this case study.

4.3 Empirical Robustness Analysis under Adversarial Perturbations

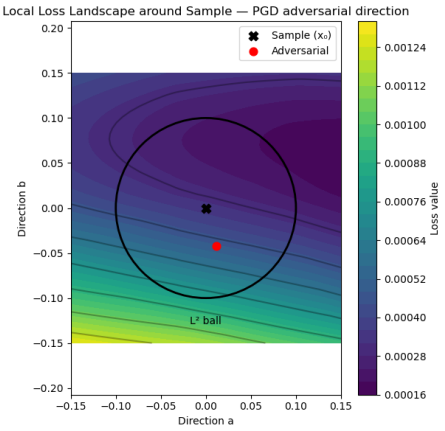
Under PGD, MSE increases with ε for both models, but the normally trained model degrades much faster; the adversarially trained model is more stable, giving a $\sim$3–4$\times$ improvement in worst-case robustness across the tested budgets (Figure 4b). The feature-level view at $\varepsilon = 0.1$ shows that relatively small, structured perturbations distributed across state features can nevertheless drive a large loss increase (Figure 4a). A complementary 2D slice of the local MSE surface around a clean input x_0 further confirms the projected gradient-ascent interpretation: the adversarial point remains within the norm ball while moving along a direction of increased loss (Figure 4c).



(a) Feature perturbations under PGD ($\varepsilon = 0.1$). Clean (solid) vs. adversarial (dashed) for a representative sample.



(b) Robustness under PGD attack: adversarial MSE vs. ε.



(c) Local loss landscape (PGD slice) around x_0.

Fig. 4: **PGD robustness and behavior.** Top: per-feature perturbations at $\varepsilon = 0.1$. Bottom-left: robustness curve across ε. Bottom-right: local loss landscape illustrating the PGD-maximised direction within the feasible region.

Lipschitz Robustness Analysis. To further assess the robustness of the learned controller, we analyse its Lipschitz behaviour by estimating the sensitivity of the network outputs to perturbations in the input state, approximating local Lipschitz constants using Jacobian-based norms. The adversarially trained controller exhibits consistently lower Lipschitz values, compared to the baseline. This indicates smoother input–output mappings. These estimates complement adversarial robustness and verification results by providing an interpretable indicator of stability.

Limitations & Future Work These results provide *empirical* robustness evidence for the chosen threat model (norm-bounded perturbations) and attack settings, but they are not a formal robustness guarantee or necessarily reflect

realistic noise. Future work can strengthen this analysis with more diverse attacks, including multi-start or momentum PGD, and with domain-consistent perturbations that better match real sensor or estimation errors.

5 Explainable AI

5.1 Adversarial Robustness with Explainability

Beyond adversarial MSE, we use Local Interpretable Model-agnostic Explanations (LIME) [25] to probe how feature reliance shifts under attack and after adversarial training by estimating local per-feature sensitivities around each sample. This complements robustness curves by revealing if the controller depends on a few brittle features or spreads importance across the state.

LIME Analysis For each test sample x, LIME fits a sparse linear surrogate in a neighbourhood of x using perturbed samples and model queries; the surrogate coefficients are treated as local feature importances. For our model, we compute LIME *per output dimension* by explaining one component $f_\theta(x)_j$ at a time. To understand the stability of explanations under adversarial attacks, we generate adversarial inputs using PGD and compare the explanations between clean and attacked versions of the same sample:

$$x_{\text{adv}} = \texttt{PGD}(x; \varepsilon, T, \alpha).$$

We set $\alpha = \varepsilon/T$ (matching our implementation default) and use ℓ_∞ PGD.

LIME Drift. To quantify how much the model's *local explanation* changes under attack, we introduce the concept of LIME drift as the ℓ_1 difference between LIME weight vectors computed on the clean sample and its adversarial counterpart, matched by feature name:

$$D_{\text{LIME}}(x) \;=\; \sum_{i=1}^{d} \big| w_i(x) - w_i(x_{\text{adv}}) \big|, \tag{14}$$

where $w_i(\cdot)$ denotes the LIME importance weight for feature i (for a fixed output dimension). A *lower* drift indicates more consistent local decision logic under adversarial perturbations.

Per-sample LIME heatmaps. Figures 8 and 9 in Appendix 8 visualise per-sample LIME feature importances (absolute weights) across multiple randomly selected test samples. Each panel shows a heatmap of size (#outputs $\times$ #features), revealing which features influence which output dimensions for that sample. The normally trained model shows *concentrated* reliance (sharp peaks) on a few features, while the adversarially trained model exhibits more *diffuse* importance across features and outputs, consistent with reduced sensitivity to bounded worst-case perturbations.

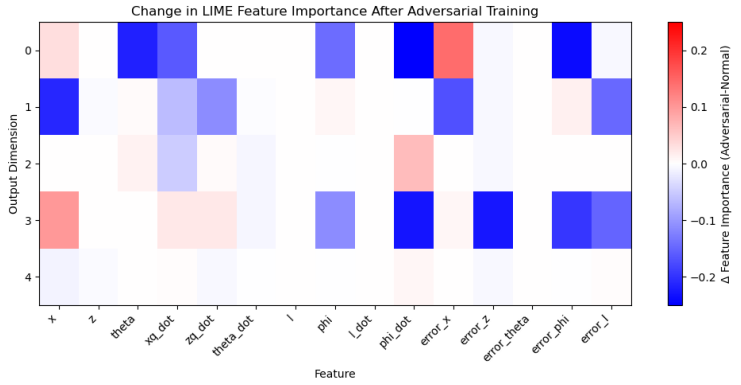


Fig. 5: **Average change in LIME importance after adversarial training.** Δw averaged over $n = 32$ random test samples, highlighting systematic shifts in feature reliance induced by adversarial training.

Change in feature reliance after adversarial training. To visualise how adversarial training alters local feature reliance, we compute difference maps

$$\Delta w \;=\; w_{\text{adv-trained}} - w_{\text{normal}}.$$

We report both per-sample and aggregated views: Figure 10 (Appendix 8) shows Δw for individual test samples, while Figure 5 (main paper) reports the mean Δw over $n = 32$ randomly selected test points. Blue indicates decreased local importance after adversarial training and red indicates increased reliance. Overall, adversarial training tends to *reduce isolated and wrong attribution spikes* and dampen large shifts in local feature weights. We include only the aggregated difference map (Figure 5) in the main paper; all per-sample heatmaps are deferred to the Appendix for completeness.

Observations. The robustness improvements seen in adversarial MSE are accompanied by improved *explanation stability*: the adversarially trained model exhibits lower mean LIME drift (Eq. 14), indicating more consistent decision logic even when inputs are attacked. Qualitatively, adversarial training encourages the controller to rely less on a narrow set of highly sensitive features and to distribute reliance across the states, which aligns with the observed increase in robustness under PGD.

Integrated Gradients We also look into Integrated Gradients (IG) [29] to answer the core question: *"Across the entire interpolation path from a baseline input to this sample, how much cumulative effect does feature i have on the output?"* We compute IG for both x and x_{adv}; paired clean vs. adversarial visualisations are provided in Appendix 8 (Figures 11–12). Large IG changes between x and x_{adv} indicate that the model's gradient-based sensitivity profile shifts under attack.

Observations. As shown in Figures 11–12, the normally trained model exhibits larger attribution changes between clean and adversarial inputs (consistent with higher LIME drift), whereas the adversarially trained model shows smaller changes and lower drift. Overall, adversarial training reduces sharp spikes and makes feature reliance/influence more consistent even under worst-case bounded perturbations, aligning with the robustness trends observed under PGD evaluation.

6 Discussion and Conclusions

This case study examined the robustness and verifiability of a neural network controller for a quadrotor with an underslung payload, highlighting both the potential and the current limitations of neural network verification tools when applied to realistic aerial robotics systems.

Our results show that controller-level formal verification using Vehicle and Marabou can successfully capture meaningful, physically interpretable properties such as bounded actuation, correct control direction, and quiet behaviour near equilibrium. Incorporating these properties into training via property-driven training measurably improved verifiability, with additional properties being formally satisfied on a benchmark problem. This demonstrates that aligning learning objectives with formal specifications can positively influence verification outcomes. However, scalability remains a key challenge: full controller-level verification of the tethered drone neural network was not tractable due to input dimensionality, dataset size, and solver constraints.

System-level reachability analysis using CORA further illustrated the difficulty of verifying closed-loop neural network–controlled systems with non-linear, coupled dynamics. While CORA provides valuable insight into finite-horizon behaviour, set explosion and computational limits restrict its applicability to higher-dimensional problems. Complementary adversarial robustness analysis supports this but does not offer formal guarantees. Explanation stability also improves, with LIME drift approximately halved and smaller Integrated Gradients under attack, indicating more consistent reliance on key features.

Our work highlights the gap between realistic cyber-physical control problems and the capabilities of current verification tooling. By combining behavioural cloning, robustness-oriented training, controller-level verification, and system-level reachability analysis, this case study provides a challenging and representative benchmark for future research. Advancing verification scalability, integrating physically meaningful specifications more directly into training, and developing tighter connections between controller-level and system-level guarantees remain important directions for future work.

References

1. Althoff, M.: An introduction to CORA 2015. In: Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems. pp. 120–151. EasyChair (December 2015). https://doi.org/10.29007/zbkv, https://easychair.org/publications/paper/xMm

2. Amer, K., Samy, M., Shaker, M., ElHelw, M.: Deep convolutional neural network based autonomous drone navigation. In: Thirteenth International Conference on Machine Vision. vol. 11605, pp. 16–24. SPIE (Jan 2021). https://doi.org/10.1117/12.2587105, https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11605/1160503/Deep-convolutional-neural-network-based-autonomous-drone-navigation/10.1117/12.2587105.full
3. Amukele, T.K., Sokoll, L.J., Pepper, D., Howard, D.P., Street, J.: Can Unmanned Aerial Systems (Drones) Be Used for the Routine Transport of Chemistry, Hematology, and Coagulation Laboratory Specimens? PLOS ONE **10**(7), e0134020 (Jul 2015). https://doi.org/10.1371/journal.pone.0134020, https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0134020, publisher: Public Library of Science
4. Brunner, G., Szebedy, B., Tanner, S., Wattenhofer, R.: The Urban Last Mile Problem: Autonomous Drone Delivery to Your Balcony. In: 2019 International Conference on Unmanned Aircraft Systems (ICUAS). pp. 1005–1012 (Jun 2019). https://doi.org/10.1109/ICUAS.2019.8798337, https://ieeexplore.ieee.org/document/8798337/, iSSN: 2575-7296
5. Crowe, W., Costales, B., Luthy, K.: Design and fabrication of a 3D-printed drone-integrated winching system. HardwareX **22**, e00642 (Jun 2025). https://doi.org/10.1016/j.ohx.2025.e00642, https://www.sciencedirect.com/science/article/pii/S2468067225000203
6. Daggitt, M., Kokke, W., Komendantskaya, E., Atkey, R., Arnaboldi, L., Slusarz, N., Casadio, M., Coke, B., Lee, J.: The vehicle tutorial: Neural network verification with vehicle. In: Narodytska, N., Amir, G., Katz, G., Isac, O. (eds.) Proceedings of the 6th Workshop on Formal Methods for ML-Enabled Autonomous Systems. Kalpa Publications in Computing, vol. 16, pp. 1–5. EasyChair (2023). https://doi.org/10.29007/5s2x, https://easychair.org/publications/paper/Rkrv
7. Enayati, S., Campbell, J.F., Li, H.: Vaccine distribution with drones for less developed countries: A case study in Vanuatu. Vaccine: X **14**, 100312 (Aug 2023). https://doi.org/10.1016/j.jvacx.2023.100312, https://www.sciencedirect.com/science/article/pii/S2590136223000530
8. Genin, D., Papusha, I., Brulé, J., Young, T., Mullins, G., Kouskoulas, Y., Wu, R., Schmidt, A.: Formal Verification of Neural Network Controllers for Collision-Free Flight. In: Bloem, R., Dimitrova, R., Fan, C., Sharygina, N. (eds.) Software Verification. pp. 147–164. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-95561-8_9
9. Glauser, W.: Blood-delivering drones saving lives in Africa and maybe soon in Canada. CMAJ : Canadian Medical Association Journal **190**(3), E88–E89 (Jan 2018). https://doi.org/10.1503/cmaj.109-5541, https://pmc.ncbi.nlm.nih.gov/articles/PMC5780273/
10. Gu, W., Valavanis, K.P., Rutherford, M.J., Rizzo, A.: UAV Model-based Flight Control with Artificial Neural Networks: A Survey. Journal of Intelligent & Robotic Systems **100**(3), 1469–1491 (Dec 2020). https://doi.org/10.1007/s10846-020-01227-8, https://doi.org/10.1007/s10846-020-01227-8
11. Haidari, L.A., Brown, S.T., Ferguson, M., Bancroft, E., Spiker, M., Wilcox, A., Ambikapathi, R., Sampath, V., Connor, D.L., Lee, B.Y.: The economic and operational value of using drones to transport vaccines. Vaccine **34**(34), 4062–4067 (Jul 2016). https://doi.org/10.1016/j.vaccine.2016.06.022, https://www.sciencedirect.com/science/article/pii/S0264410X16304352

12. Ho, F.D., Lomax, J.C., Mueller, M.: Optimizing Parachute Payload Delivery Through Wind Gust Analysis. In: AIAA SCITECH 2025 Forum. American Institute of Aeronautics and Astronautics, Orlando, FL (Jan 2025). https://doi.org/10.2514/6.2025-2348, https://arc.aiaa.org/doi/10.2514/6.2025-2348
13. Hughes, K., King, K., Hickman, S., Rannow, A., Freisinger, G., Huntoon, S., Kuhlwein, E., Rooney, B.: Autonomous Aerial Supply Delivery: Parachute Release and Impact Study. American Society of Mechanical Engineers Digital Collection (Feb 2023). https://doi.org/10.1115/IMECE2022-93895, https://dx.doi.org/10.1115/IMECE2022-93895
14. Iqbal, U., Shah, S., Sadiq, M.: Servo actuated payload carry and drop mechanism for unmanned helicopter. pp. 1–6 (Oct 2016). https://doi.org/10.1109/ICET.2016.7813221
15. Kessler, C., Komendantskaya, E., Casadio, M., Viola, I.M., Flinkow, T., Othman, A.A., Malhotra, A., McPherson, R.: Neural Network Verification for Gliding Drone Control: A Case Study (May 2025). https://doi.org/10.48550/arXiv.2505.00622, http://arxiv.org/abs/2505.00622, arXiv:2505.00622 [cs]
16. Ling, G., Draghic, N.: Aerial drones for blood delivery. Transfusion **59**(S2), 1608–1611 (Apr 2019). https://doi.org/10.1111/trf.15195
17. Lopez, D.M., Althoff, M., Forets, M., Johnson, T.T., Ladner, T., Schilling, C.: ARCH-COMP23 category report: 10th International Workshop on Applied Verification of Continuous and Hybrid Systems. 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23) pp. 89–125 (2023). https://doi.org/10.29007/x38n, publisher: EasyChair
18. Maheshwar, E.M., Ibrahim, R., Nithiyanantham, K.K.: Development of a Payload-Dropping Quadcopter Using Landing Gears with Electronic Servo. In: Tambe, P., Huang, P., Jhavar, S. (eds.) Advances in Mechanical Engineering and Material Science. pp. 209–226. Springer Nature, Singapore (2024). https://doi.org/10.1007/978-981-99-5613-5_17
19. Mazur, M., Borucka, A.: Commercial Drone Deliveries: Evolution, Applications, and Market Dynamics. In: 2025 13th International Conference on Traffic and Logistic Engineering (ICTLE). pp. 336–340 (Aug 2025). https://doi.org/10.1109/ICTLE67020.2025.11203681, https://ieeexplore.ieee.org/document/11203681/
20. Musa, A.A., Hussaini, A., Liao, W., Liang, F., Yu, W.: Deep Neural Networks for Spatial-Temporal Cyber-Physical Systems: A Survey. Future Internet **15**(6), 199 (Jun 2023). https://doi.org/10.3390/fi15060199, https://www.mdpi.com/1999-5903/15/6/199, publisher: Multidisciplinary Digital Publishing Institute
21. Nisingizwe, M.P., Ndishimye, P., Swaibu, K., Nshimiyimana, L., Karame, P., Dushimiyimana, V., Musabyimana, J.P., Musanabaganwa, C., Nsanzimana, S., Law, M.R.: Effect of unmanned aerial vehicle (drone) delivery on blood product delivery time and wastage in Rwanda: a retrospective, cross-sectional study and time series analysis. The Lancet. Global Health **10**(4), e564–e569 (Apr 2022). https://doi.org/10.1016/S2214-109X(22)00048-1
22. Ospina-Fadul, M.J., Kremer, P., Stevens, S.E., Haruna, F., Okoh-Owusu, M., Sarpong, G.K., Osei-Kwakye, K., Joshua, B., Sakyi, O.: Cost-Effectiveness of Aerial Logistics for Immunization: A Model-Based Evaluation of Centralized Storage and Drone Delivery of Vaccines in Ghana Using Empirical Data. Value in Health Regional Issues **46**, 101066 (Mar 2025). https://doi.org/10.1016/j.vhri.2024.101066, https://www.sciencedirect.com/science/article/pii/S2212109924000992

23. Pillai, A.V., K, L.: Optimizing UAV Payload Delivery Mechanisms: A Systematic Analysis of Mechanical Parameters Affecting Descent Rates. In: 2025 International Conference on Emerging Technology in Autonomous Aerial Vehicles (ETAAV). pp. 1–6 (Aug 2025). https://doi.org/10.1109/ETAAV66793.2025.11212991, https://ieeexplore.ieee.org/document/11212991
24. Poulsen, E., Rysgaard, S., Hansen, K., Karlsson, N.B.: Uncrewed aerial vehicle with onboard winch system for rapid, cost-effective, and safe oceanographic profiling in hazardous and inaccessible areas. HardwareX **18**, e00518 (Jun 2024). https://doi.org/10.1016/j.ohx.2024.e00518, https://www.sciencedirect.com/science/article/pii/S2468067224000129
25. Ribeiro, M.T., Singh, S., Guestrin, C.: " why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1135–1144 (2016)
26. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why Should I Trust You?": Explaining the Predictions of Any Classifier (Aug 2016). https://doi.org/10.48550/arXiv.1602.04938, http://arxiv.org/abs/1602.04938, arXiv:1602.04938 [cs]
27. Seshia, S.A., Desai, A., Dreossi, T., Fremont, D.J., Ghosh, S., Kim, E., Shivakumar, S., Vazquez-Chanlatte, M., Yue, X.: Formal Specification for Deep Neural Networks. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis. pp. 20–34. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_2
28. Sun, X., Khedr, H., Shoukry, Y.: Formal verification of neural network controlled autonomous systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 147–156. HSCC '19, Association for Computing Machinery, New York, NY, USA (Apr 2019). https://doi.org/10.1145/3302504.3311802, https://dl.acm.org/doi/10.1145/3302504.3311802
29. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: International conference on machine learning. pp. 3319–3328. PMLR (2017)
30. Wu, H., Isac, O., Zeljić, A., Tagomori, T., Daggitt, M., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., et al.: Marabou 2.0: a versatile formal analyzer of neural networks. In: International Conference on Computer Aided Verification. pp. 249–264. Springer (2024)
31. Yu, G., Reis, J., Silvestre, C.: Quadrotor Neural Network Adaptive Control: Design and Experimental Validation. IEEE Robotics and Automation Letters **8**(5), 2574–2581 (May 2023). https://doi.org/10.1109/LRA.2023.3254450, https://ieeexplore.ieee.org/document/10064027/

7 Appendix

7.1 Numerical Simulation Framework

The system of equations derived in Section 2 describes a coupled set of nonlinear, second-order differential equations governing the motion of the quadcopter drone and tethered payload. Closed-form solutions are not available; therefore, the dynamics are integrated numerically over discrete timesteps using a fourth-order Runge–Kutta (RK4) scheme with timestep Δt.

At each simulation timestep, the following operations are performed:

1. Evaluate the current state vector $\boldsymbol{s}(t)$.
2. Compute the state derivatives from the continuous-time dynamics:

$$\dot{\boldsymbol{s}} = f(\boldsymbol{s}, \boldsymbol{u}, \boldsymbol{p}),$$

   where $\boldsymbol{u}$ denotes the control inputs and $\boldsymbol{p}$ represents the physical parameters loaded from a configuration file (`config.yaml`).
3. Advance the state using RK4 integration:

$$\boldsymbol{s}(t + \Delta t) = \boldsymbol{s}(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

   with

$$\begin{aligned} k_1 &= f(\boldsymbol{s}_t), \\ k_2 &= f(\boldsymbol{s}_t + \tfrac{1}{2}\Delta t k_1), \\ k_3 &= f(\boldsymbol{s}_t + \tfrac{1}{2}\Delta t k_2), \\ k_4 &= f(\boldsymbol{s}_t + \Delta t k_3). \end{aligned}$$

4. Resolve the algebraic coupling between tether tension Q and quadcopter acceleration using a quasi-static fixed-point iteration:

$$Q^{(i+1)} = m_p\Big(g\cos\phi + \boldsymbol{a}_q \cdot \boldsymbol{u} - (\ddot{\ell} - \ell\dot{\phi}^2)\Big),$$

   where $\boldsymbol{a}_q = (\ddot{x}_q, \ddot{z}_q)^\top$. For $\Delta t < 0.05$ s, convergence is typically achieved within one or two iterations.

The simulation is implemented in a modular Python framework:

- `dynamics.py`: continuous-time dynamics and force computations,
- `simulation.py`: RK4 integration loop and data logging,
- `pid_control.py`: expert PID controller implementation.

This simulation environment serves both as a validation tool for the expert controller and as the data-generation engine for training the neural network.

7.2 Free-Body Diagram

Fig. 6 illustrates the free-body diagram of the quadcopter and the payload.

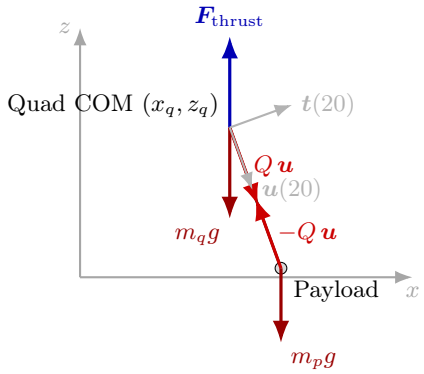


Fig. 6: Free-body diagram showing the forces acting on the quadcopter and the tethered payload.

7.3 Tether Geometry and Kinematic Derivations

We consider planar motion of a quadcopter with a payload suspended beneath it by a massless, inextensible tether of variable length ℓ. The tether configuration is parameterised by the swing angle ϕ, measured from the vertical. Unit vectors aligned with and perpendicular to the tether are defined as

$$\boldsymbol{u}(\phi) = \begin{bmatrix} \sin\phi \\ -\cos\phi \end{bmatrix}, \qquad \boldsymbol{t}(\phi) = \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix}.$$

The payload position is expressed as

$$\boldsymbol{p}_p = \begin{bmatrix} x_q \\ z_q \end{bmatrix} + \ell\,\boldsymbol{u}(\phi),$$

and differentiation yields

$$\ddot{\boldsymbol{p}}_p = \begin{bmatrix} \ddot{x}_q \\ \ddot{z}_q \end{bmatrix} + (\ddot{\ell} - \ell\dot{\phi}^2)\,\boldsymbol{u} + (2\dot{\ell}\dot{\phi} + \ell\ddot{\phi})\,\boldsymbol{t}.$$

These expressions are used to derive the payload swing and tether tension equations presented in the main text.

7.4 PID Controller Implementation Details

Cascaded Control Structure The PID controllers are arranged in a cascaded architecture, as shown in Fig. 7. Outer-loop controllers generate reference commands for inner-loop controllers, which stabilise the quadcopter attitude and vertical motion.

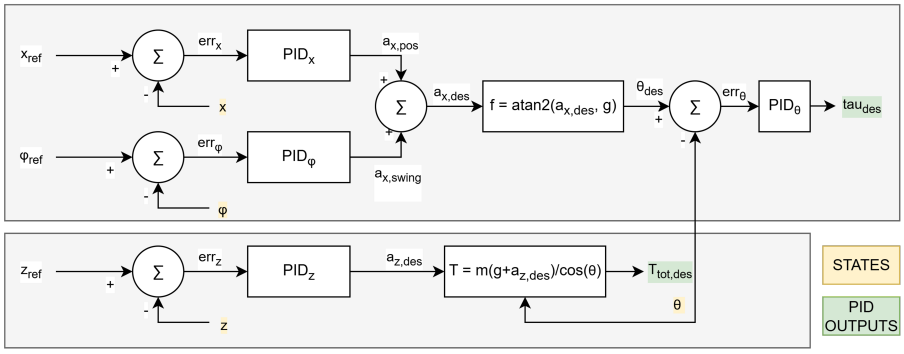


Fig. 7: Cascaded PID controller diagram

Control Allocation The translational and rotational control outputs are mapped to individual rotor thrusts using a standard mixing matrix:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = M \begin{bmatrix} u_z \\ u_\theta \end{bmatrix},$$

where M is determined by the quadcopter geometry and rotor configuration.

Anti-Windup and Saturation Handling To ensure stable behaviour under actuator limits, saturation bounds are imposed on all control outputs. Integral windup is mitigated by freezing the integral term whenever saturation occurs in a direction that would further increase the error:

$$(u_i > u_{\max} \wedge e_i > 0) \ \vee \ (u_i < u_{\min} \wedge e_i < 0) \implies \dot{I}_i = 0$$

This prevents integrator accumulation during saturation and enables smooth recovery once the system re-enters the linear operating regime.

7.5 Vehicle Specification of Formal Properties

This appendix provides the complete Vehicle specification of the formal properties evaluated in Section 4.1.

Initialisation and helper methods (Listing 1.1). We treat the network input as an n-dimensional real vector `State` containing the relevant state components and tracking errors, and the network output as a d-dimensional real vector `Ctrl` containing the control commands (system agnostic specification). The predicate `validState` encodes the bounded input domain $\mathcal{X}$ by enforcing component-wise lower/upper bounds (`minimumInputValues`, `maximumInputValues`).

```
type State = Tensor Real [n] -- n-dim vector of states+errors
type Ctrl  = Tensor Real [d] -- d-dim vector of control
    outputs

-- Absolute value
abs : Real -> Real
abs r = if r >= 0 then r else -r

-- Input domain (bounded quantification)
minimumInputValues : State
maximumInputValues : State

validState : State -> Bool
validState x = forall i . minimumInputValues ! i <= x ! i <=
    maximumInputValues ! i
```

Listing 1.1: System-agnostic VEHICLE specification initialisation.

Network definition and normalisation (Listing 1.2). The function `normalise` applies the same per-feature affine normalisation used during training, using the vectors named `meanScalingValues` (μ) and `stdDevScalingValues` (σ). The network itself is declared as `@network ctrlNet : State -> Ctrl`, and `applyCtrl` wraps the network together with the normalisation.

```
-- Normalisation
@parameter
meanScalingValues   : State    -- mu (per-input-feature mean)
@parameter
stdDevScalingValues : State    -- sd (per-input-feature std)

normalise : State -> State
normalise x =
  foreach i . (x ! i - meanScalingValues ! i) /
      stdDevScalingValues ! i

-- Network definition
@network
ctrlNet : State -> Ctrl

applyCtrl : State -> Ctrl
applyCtrl x = ctrlNet ( normalise x )
```

Listing 1.2: Normalisation and definition of neural network in VEHICLE.

Constants used by properties (Listing 1.3). `u_min` and `u_max` are vectors describing the control bounds; `m` is the margin for Property P1.2; `epsilon` is the "far-from-reference" threshold for Property P2; and the vectors `eps_eq` and `u_quiet` define the equilibrium neighbourhood and allowed small-control region

for Property P3. The elements of `eps_eq` and `u_quiet` should always be positive to compare with absolute state and control values.

```
-- Constants used by properties
@parameter
u_min : Ctrl     -- actuator lower bounds (vector)
@parameter
u_max : Ctrl     -- actuator upper bounds (vector)

@parameter
m : Real         -- margin for P1.2 (m > 0)

@parameter
epsilon : Real   -- "far from reference" threshold (>0) for P2
@parameter
eps_eq  : State  -- "near equilibrium" threshold vector for P3
@parameter
u_quiet : Ctrl   -- quiet-control bound vector for P3
```

Listing 1.3: VEHICLE constants used by formal properties.

Formal properties. Properties P1–P3 are described through Listings 1.4, 1.5 and 1.6, respectively.

```
-- Property P1: Global Saturation Safety
@property
P1_global_saturation : Bool
P1_global_saturation =
  forall x .
    validState x =>
      forall i . u_min ! i <= (applyCtrl x) ! i <= u_max ! i

-- Property P1.2: Saturation Safety with Margin
@property
P1_2_saturation_margin : Bool
P1_2_saturation_margin =
  forall x .
    validState x =>
      forall i . (u_min ! i - m) <= (applyCtrl x) ! i <= (
          u_max ! i + m)
```

Listing 1.4: VEHICLE specification for Properties P1 and P1.2.

Property P2 (Listing 1.5) affects one control channel. Therefore, if the property is to be applied over several control axes, different instances of the same property should be defined. Here, `e_idx` and `u_idx` are the corresponding state and control output indexes representing the error and control values of the relevant channel. For example, consider Property P2 wants to enforce that if the drone is too far left from the setpoint (the horizontal error e_x is largely negative $e_x \leq -\epsilon$), then the control output corresponding to the x-axis (u_x) should push

to the right ($u_x > 0$). This corresponds to the x-channel, and `e_idx` and `u_idx` should be set as the indexes corresponding to e_x and u_x in the state and value vectors.

```
-- Property P2: Correct Control Direction at Extremes (
    example on one channel)
@property
P2_correct_direction : Bool
P2_correct_direction =
  forall x .
    validState x =>
      ( (x ! e_idx) >  epsilon => (applyCtrl x) ! u_idx < 0 )
          and
      ( (x ! e_idx) < -epsilon => (applyCtrl x) ! u_idx > 0 )
```

Listing 1.5: VEHICLE specification for Property P2.

```
-- Property P3: Quiet Control Near Equilibrium
nearEquilibrium : State -> Bool
nearEquilibrium x = forall i . abs (x ! i) <= eps_eq ! i

@property
P3_quiet_near_equilibrium : Bool
P3_quiet_near_equilibrium =
  forall x .
    validState x and nearEquilibrium x =>
      forall j . abs ((applyCtrl x) ! j) <= u_quiet ! j
```

Listing 1.6: VEHICLE specification for Property P3.

7.6 CORA property specification and reachability setup

```
% Closed-loop nonlinear system (PID embedded in dynamics)
nX = 13;   % 10 plant states + 3 PID integral states
nU = 1;    % no external inputs (closed-loop)
fCL   = @(x,u) quadPayloadPID_ode_lin(x,u,p);
sysCL = nonlinearSys(fCL, nX, nU);

% Initial condition with bounded uncertainty
x0_nom  = [0; 5; 0; 0; 0; 0; 1; 0; 0; 0];
e_int0  = [0; 0; 0];
x0_full = [x0_nom; e_int0];

eps0 = 0.01;
params.R0 = zonotope(interval(x0_full - eps0, x0_full + eps0)
    );

% Reachability parameters
params.tFinal = 4;
params.U      = zonotope(0);

options.timeStep      = 0.001;
options.taylorTerms   = 4;
options.zonotopeOrder = 20;
options.alg           = 'poly';
options.tensorOrder   = 3;
options.errorOrder    = 3;
options.intermediateOrder = 3;

% Specification sets
INF = 1e6;
lb_goal = -INF*ones(nX,1);  ub_goal = INF*ones(nX,1);
lb_goal(2) = 2.0;           ub_goal(2) = 10.0;
goalSet = interval(lb_goal, ub_goal);

lb_avoid = -INF*ones(nX,1); ub_avoid = INF*ones(nX,1);
lb_avoid(2) = 0.0;          ub_avoid(2) = 1.0;
avoidSet = interval(lb_avoid, ub_avoid);

% Time-bounded specifications
spec1 = specification(goalSet,  'safeSet',   interval(params.
    tFinal-1, params.tFinal));
spec2 = specification(avoidSet, 'unsafeSet', interval(0,
    params.tFinal));
spec  = add(spec1, spec2);

% Reachability verification
[RCL, resCL] = reach(sysCL, params, options, spec);
```

Listing 1.7: Closed-loop reachability specification in CORA

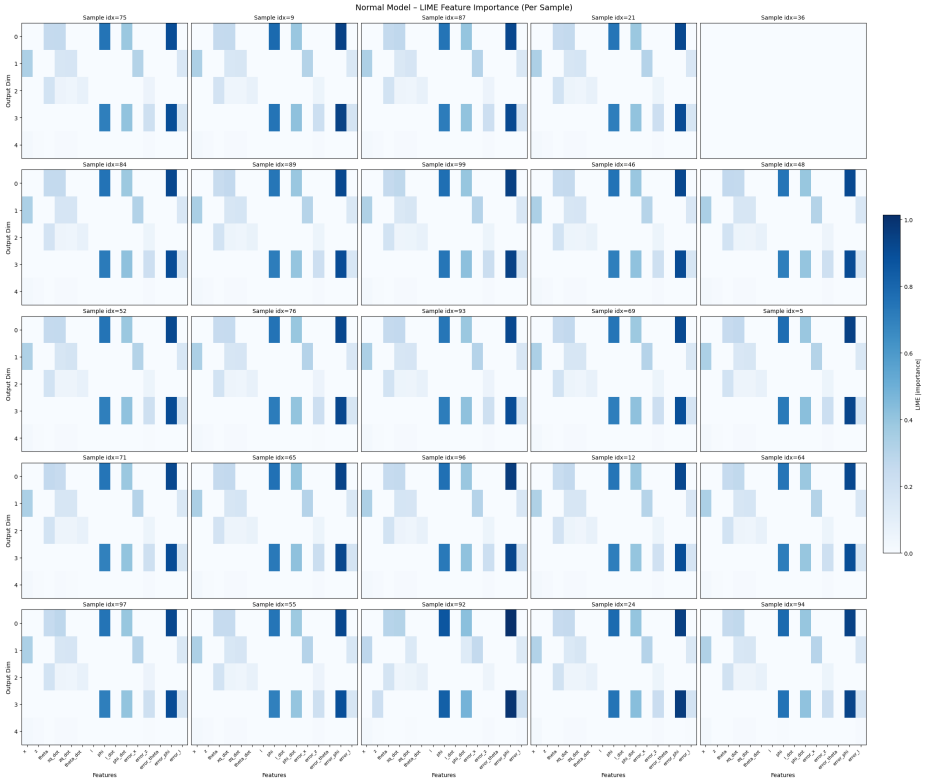


Fig. 8: **Normal model: LIME feature importance per sample.** Each panel shows a heatmap of absolute LIME weights (#outputs × #features) for a randomly selected test sample. Darker cells indicate stronger local reliance.

8 Additional Explainability Visualizations

This appendix provides supplementary LIME and Integrated Gradients (IG) visualisations referenced in the main text (Section 5.1). We include per-sample LIME heatmaps for the normally trained and adversarially trained controllers, as well as per-sample difference maps that highlight how local feature reliance changes after adversarial training. We also report representative IG bar plots (clean vs. adversarial) together with the corresponding LIME drift values to illustrate stability under attack.



Fig. 9: **Adversarially trained model: LIME feature importance per sample.** Compared to the normal model, feature importance is generally less spiky, suggesting reduced reliance on brittle feature pathways.

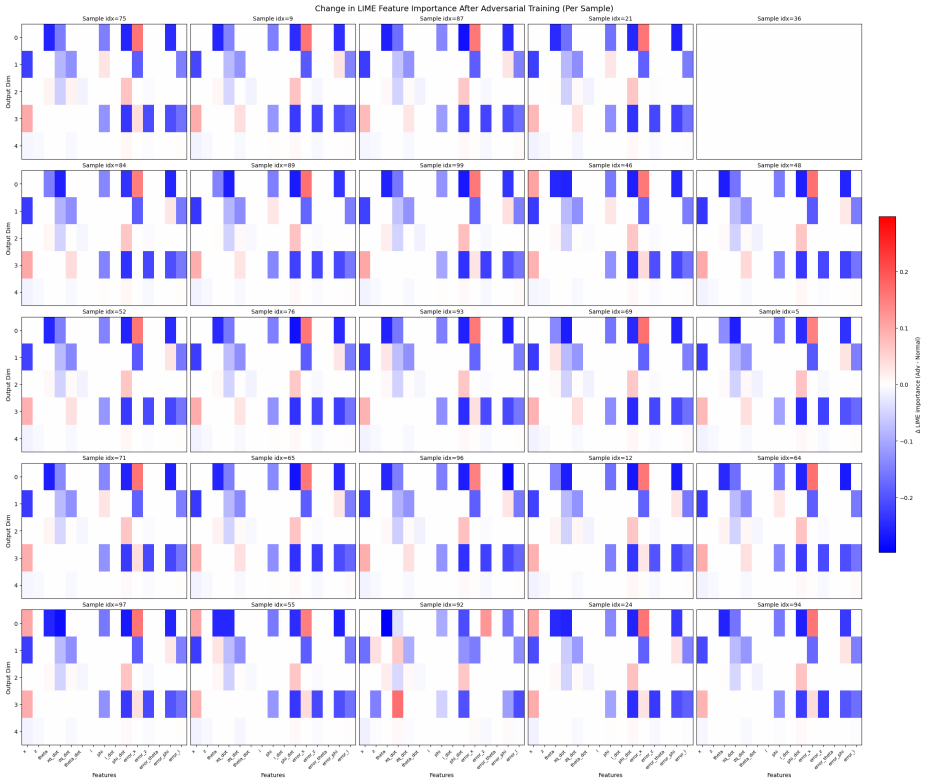


Fig. 10: **Per-sample change in LIME importance after adversarial training.** Heatmaps show $\Delta w = w_{\text{adv-trained}} - w_{\text{normal}}$ (per output $\times$ feature). Blue indicates decreased reliance after adversarial training; red indicates increased reliance.

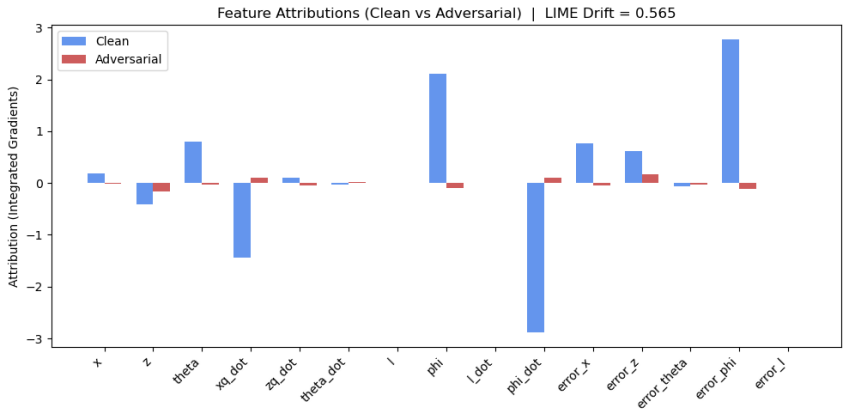


Fig. 11: **Normal model: Integrated Gradients (clean vs. adversarial) with LIME drift.** The paired bars show IG attributions per feature for output j=0 on the clean input x and the adversarial input x_{adv}. The reported LIME drift (0.565) indicates a larger change in local explanation under attack.

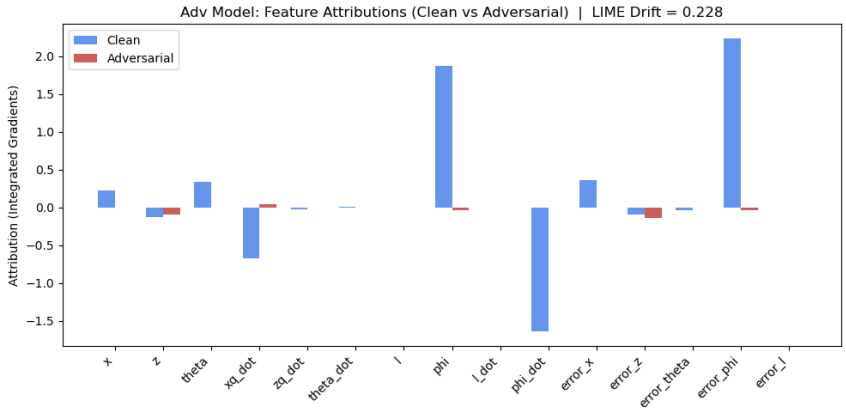


Fig. 12: **Adversarially trained model: Integrated Gradients (clean vs. adversarial) with LIME drift.** Compared to the normal model, both the IG profile shift and the LIME drift are smaller (0.228), indicating more stable feature reliance under adversarial perturbations.