TNODEV: Toolbox for Neural ODE Verification

Anonymous authors
Paper under double-blind review

Abstract

Neural ordinary differential equations (neural ODE) gained attention in safety critical settings such as continuous-time controllers for cyber-physical systems and classifiers integrated into automated decision pipelines, raising the question whether their behavior can be formally verified. Existing tools dedicated to neural ODE provide only a single reachability call without iterative input-set refinement, limiting the precision of their verdicts to whatever one reachability call can deliver. We present TNODEV, the first formal verifier for neural ODE that integrates a falsification checker, a fast interval-based reachability backend based on continuous-time mixed monotonicity, a verification and refinement loop with three input-set splitting heuristics, and a parallel scheduler in a single end-to-end pipeline. TNODEV supports safe-set inclusion verification on pure neural ODE, neural ODE in closed loop with a neural network controller and general neural ODE (GNODE), with the safe set specified either as an interval or as the half-space intersection induced by a target classification label. We evaluate TNODEV on a range of benchmarks across safe-set inclusion and classification-robustness properties, including a direct reachability comparison against NNV 2.0 and CORA and a verification comparison against NNV 2.0 on MNIST general neural ODE classifiers.

1 Introduction

Neural ordinary differential equations (neural ODE) (Rico-Martínez et al., 1992; Chen et al., 2018) parameterize the derivative of a hidden state by a neural network, so that the forward pass is the solution of an initial value problem rather than the application of a finite stack of layers. Concretely, we consider the autonomous neural ODE

$$\dot{x}(t) \;=\; \frac{\mathrm{d}x(t)}{\mathrm{d}t} \;=\; f(x(t)), \qquad x(0) = u, \tag{1}$$

in which the right-hand side f is parameterized by a neural network. This continuous-time formulation has been shown to be effective across a range of tasks such as time series modeling (Rubanova et al., 2019), generative modeling (Grathwohl et al., 2018), and as continuous-depth substitutes for residual networks Chen et al. (2018). As neural ODE are finding applications in higher-stakes settings, including the controllers of safety critical cyber-physical systems (Lechner et al., 2020; Xiao et al., 2023) and classifiers integrated into automated decision pipelines (Moon et al., 2022; Qian et al., 2021), the question of whether their behavior can be formally verified becomes increasingly relevant.

Formal verification of neural ODE differs from formal verification of standard neural networks in two ways. First, the forward pass is the solution of a continuous-time initial value problem, which means that bounding the reachable set requires propagating an input set through the flow of the differential equation rather than through a finite stack of discrete layers. Second, the trade-offs of standard neural network verification, such as the choice of set representation, the management of branching in non-piecewise-linear activations, and the use of input-set refinement to recover precision lost to over-approximation, all carry over to the neural ODE setting but interact differently with the continuous-time dynamics. As a result, the available verification tools for neural ODE are much less mature than their neural network counterparts, and existing tools differ substantially in the type of guarantees they provide and in the verification workflow they expose to the user.

A look at the current literature (Table 1) shows that all existing tools dedicated to neural ODE contain only a reachability step, returning a single sound over-approximation of the reachable set per query. Currently, no tool implements the full verification pipeline that has become standard in the neural network verification literature (Liu et al., 2021; Bunel et al., 2020; Wang et al., 2021). Such a pipeline consists of a falsification pass that searches for explicit counter-examples, a reachability call that computes a sound over-approximation, and an iterative input-set refinement loop that splits inconclusive cells and re-runs the reachability call on each sub-cell until the property is certified or a maximum number of refinement iterations is reached. The closest existing tool to a full pipeline is NNV 2.0 (Lopez et al., 2023), which checks safety specifications on the output of a single reachability call but does not perform input-set refinement. Since current neural ODE verifiers omit this refinement step, their verification decision is based on a single reachability call, whose over-approximation might be too conservative to guarantee the satisfaction of the desired properties, and no attempt is made to tighten it in the hope of improving the verification result.

In this paper, we present **TNODEV**, a formal verifier for neural ODE that combines in a single end-to-end pipeline and for the first time in the neural ODE literature: a falsification module, a reachability backend, an iterative input-set refinement loop with three splitting heuristics, and a parallel scheduler. TNODEV uses the continuous-time mixed-monotonicity (CTMM) reachability method (Meyer et al., 2021) as its default reachability backend, chosen for its low per-call cost which makes it well suited to a refinement loop that re-computes reachability on many sub-cells. The verifier returns one of three verdicts (`SAFE`, `FALSIFIED`, or `UNKNOWN`) and currently supports safe-set inclusion verification on three model classes (pure neural ODE, neural ODE in closed loop with a neural network controller, general neural ODE), with the safe set specified either as an interval or as the half-space intersection induced by a target classification label. The architecture is modular and can be extended to additional reachability analysis methods, model classes, and specification types.

The paper is organized as follows. Section 2 reviews existing tools for neural ODE reachability and verification, and positions TNODEV against them. Section 3 introduces the necessary preliminaries on neural ODE reachability and the CTMM reachability method. Section 4 describes TNODEV verifier architecture, including its falsification module, specification interface, verification and refinement loop, three refinement heuristics, and the parallel scheduler. Section 5 reports the experimental evaluation in three parts: a direct reachability comparison against NNV 2.0 and CORA, a verification evaluation on six safe-set inclusion benchmarks across two model classes, and a classification robustness evaluation on MNIST GNODE classifiers. Section 6 concludes the paper and discusses future directions.

2 Related work

The verification of neural ODE is a recent research direction, and the available choice of tools is much narrower than the current available tools for neural networks. Existing tools differ along three main axes: (i) whether they are designed for neural ODE specifically or adapted to them from a more general setting; (ii) the type of guarantees they provide, i.e., deterministic (sound) or probabilistic; (iii) and whether they use only reachability analysis as the main primitive or a complete verifier pipeline that includes a falsification module and input-set refinement. Table 1 summarizes the main tools and techniques currently available for the reachability analysis and verification of neural ODE.

The first work on neural ODE reachability is the *Stochastic Lagrangian Reachability* (SLR) method of Grunbacher et al. (2021), which combines Lipschitz-based bounds on the flow map with a stochastic optimization procedure to produce confidence intervals on the reachable set. This reachability method was subsequently extended to longer time horizons in the *GoTube* tool (Gruenbacher et al., 2022). Both methods provide probabilistic rather than deterministic guarantees, and the distinction is important for the user, as a deterministic reachability method returns a set that is mathematically proven to contain every trajectory of the system, while a probabilistic reachability method returns a set that contains every trajectory only with a user-specified confidence level, leaving a non-zero residual probability that a trajectory ends outside the reported set. Such stochastic approaches can be more appropriate for runtime monitoring and statistical assurance use cases but not for safety-critical systems, where any non-zero probability of missing an unsafe behavior is unacceptable.

The first deterministic tool dedicated to neural ODE is *NNVODE* (Manzanas Lopez et al., 2022), which introduces a *General Neural ODE* (GNODE) class combining discrete neural network layers (fully connected, convolutional) with continuous ODE layers. *NNVODE* is built as an extension of the neural network verification tool *NNV* (Tran et al., 2020), it handles the discrete neural network layers with *NNV*'s star set-based reachability analysis (Tran et al., 2019) and the continuous ODE block with the zonotope-based reachability backend of CORA (Althoff, 2015). This work was later integrated into *NNV 2.0* tool (Lopez et al., 2023), which is the closest existing tool to TNODEV in scope. However, the main difference from TNODEV is that *NNV 2.0* performs only one star-set reachability call per query without any input-set refinement, so the precision of its verdict is limited only to a single reachability analysis call. More recently, *ModelVerification.jl* (Wei et al., 2025) a Julia-based toolbox building on the earlier *NeuralVerification.jl* (Liu et al., 2021), provides deterministic guarantees through reachability analysis and lists neural ODE among its supported network architectures alongside feed-forward networks, convolutional networks, and ResNet.

Another line of work uses general-purpose reachability tools that were not originally designed for neural ODE but can be adapted to them. *CORA* (Althoff, 2015) is a widely-used reachability library for linear, nonlinear, and hybrid dynamical systems based on zonotopes and their generalizations (matrix zonotopes, polynomial zonotopes). Its `nonlinearSys` backend can be applied to a neural ODE by treating it as a generic nonlinear vector field. Along these lines, Liang et al. (2024) combine CORA's zonotope-based reachability with a set-boundary analysis that exploits the homeomorphism property of invertible neural networks (including neural ODE) to propagate only the boundary of the input set rather than the entire set. The most recent addition to the literature on neural ODE reachability analysis is Sayed et al. (2025b), relying on the continuous-time mixed monotonicity (CTMM) method from *TIRA* (Meyer et al., 2019), declined in three versions: computing the final reachable set in a single integration step; incrementally with intermediate time steps; and combined with a boundary-based approach. However, similar to CORA, it only contains a reachability step. TNODEV uses the CTMM method as its default reachability backend and wraps it as a full neural ODE verifier.

A different approach to neural ODE verification proceeds indirectly, by establishing formal relationships between neural networks and neural ODE. Sayed et al. (2025a) establish a formal error bound between a neural ODE and the ResNet obtained by treating one Euler step of the ODE as a single residual block, so that any specification certified on one of the two models can be transferred to the other. This approach is complementary to the direct reachability analysis tools for neural ODE discussed above, as it allows for re-using any of the available neural network verification tools.

Table 1: Tools and techniques that handle neural ODE reachability and/or verification. "Designed for nODE" indicates whether the tool was developed specifically for neural ODE or adapted from a more general setting. The columns "Use for nODE" and "Iterative refinement for nODE" describe each tool's behavior *when applied to neural ODE* only, as some tools support broader functionality on standard neural networks.

Tool/Technique	**Set Representation**	**Guarantee**	**Designed for nODE**	**Use for nODE**	**Iterative refinement for nODE**
SLR (Grunbacher et al., 2021)	Lipschitz balls	Probabilistic	✓	Reachability	no
GoTube (Gruenbacher et al., 2022)	Lipschitz balls	Probabilistic	✓	Reachability	no
NNVODE (Manzanas Lopez et al., 2022)	Star set + zonotopes	Deterministic	✓	Reachability	no
NNV 2.0 (Lopez et al., 2023)	Star set + zonotopes	Deterministic	adapted	Verification	no
ModelVerification.jl (Wei et al., 2025)	Multiple	Deterministic	adapted	Verification	no
CORA (Althoff, 2015)	Zonotopes	Deterministic	adapted	Reachability	no
Liang et al. (2024) + CORA	Zonotopes	Deterministic	adapted	Reachability	no
TIRA + CTMM (Sayed et al., 2025b)	Intervals	Deterministic	✓	Reachability	no
Formal relationship of Sayed et al. (2025a)	Zonotopes	Deterministic	✓	Verification	not applicable
TNODEV (ours)	Intervals	Deterministic	✓	Verification	yes

3 Preliminaries

3.1 Neural ODE Reachability

Definition 1 (Neural ODE Reachability)**.** *Given an initial set $\mathcal{X}_{in} \subseteq \mathbb{R}^n$ and final time t_f for the neural ODE, the set of reachable outputs is defined as:*

$$\mathcal{R}(\mathcal{X}_{in}) \;=\; \{\, \Phi(t_f, u) \;\mid\; u \in \mathcal{X}_{in} \,\},$$

where $\Phi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is the solution map of the initial value problem associated with equation (1), such that $\Phi(t_f, u)$ is the state reached at time t_f from the initial condition $x(0) = u$.

Since $\mathcal{R}(\mathcal{X}_{in})$ cannot be computed exactly, we rely on an over-approximation $\Omega(\mathcal{X}_{in})$ such that $\mathcal{R}(\mathcal{X}_{in}) \subseteq \Omega(\mathcal{X}_{in})$.

Definition 1 characterizes the reachable set at the single time instant t_f. For the construction of the CTMM decomposition function in Section 3.2, we need to initially obtain some bounds on the union of all the reachable sets over the entire time interval $[0, t_f]$, which we refer to as the *reachable tube*.

Definition 2 (Neural ODE Reachable Tube)**.** *Given an initial set $\mathcal{X}_{in} \subseteq \mathbb{R}^n$ and a final time t_f for the neural ODE, the reachable tube is defined as:*

$$\mathcal{R}^{tube}(\mathcal{X}_{in}) \;=\; \bigcup_{t \in [0, t_f]} \{\, \Phi(t, u) \mid\; u \in \mathcal{X}_{in} \,\}.$$

This reachable tube contains every possible reachable state of the system starting from $\mathcal{X}_{in}$ over the time range $[0, t_f]$, and therefore provides a domain over which the Jacobian of f can be bounded.

3.2 Reachability via Continuous-Time Mixed Monotonicity

The reachability component at the core of TNODEV relies on CTMM (Meyer et al., 2021; Coogan, 2020), a property of dynamical systems that allows the original system to be embedded into a higher-dimensional monotone system on which interval bounds can be propagated efficiently.

Given the neural ODE equation (1) and an interval initial set $\mathcal{X}_{in} = [\underline{x}_0, \overline{x}_0] \subseteq \mathbb{R}^n$, the procedure can be carried out in three steps.

Step 1: Reachable tube and Jacobian bounding. Compute an interval over-approximation $\Omega^{\text{tube}}(\mathcal{X}_{in}) \supseteq \mathcal{R}^{\text{tube}}(\mathcal{X}_{in})$ of the reachable tube (Definition 2) using an external reachability tool for neural ODE, e.g. CORA (Althoff, 2015) in our case. From this over-approximation, derive interval bounds $[\underline{J}_x, \overline{J}_x]$ on the Jacobian $J_x(x) = \partial f / \partial x$ for all $x \in \Omega^{\text{tube}}(\mathcal{X}_{in})$, using interval arithmetic applied to the Jacobian of the neural network. Lipschitz continuity of f guarantees that these bounds exist, and they are the only condition required for the next step.

Step 2: Decomposition function. Construct a decomposition function $g : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ satisfying $g(x, x) = f(x)$ for all x, such that g is increasing in its first argument (off-diagonally) and decreasing in its second argument. This is achieved by a sign-stabilization procedure that shifts each off-diagonal entry of the Jacobian into a constant-sign half-plane using a shifting matrix L_x derived from $[\underline{J}_x, \overline{J}_x]$ (Sayed et al., 2025b).

Step 3: Embedded simulation. Embed the neural ODE into the $2n$-dimensional monotone system

$$\begin{bmatrix} \dot{x} \\ \dot{\hat{x}} \end{bmatrix} \;=\; \begin{bmatrix} g(x, \hat{x}) \\ g(\hat{x}, x) \end{bmatrix}, \qquad \begin{bmatrix} x(0) \\ \hat{x}(0) \end{bmatrix} \;=\; \begin{bmatrix} \underline{x}_0 \\ \overline{x}_0 \end{bmatrix}, \tag{2}$$

and solve it once with a numerical ODE solver from $t = 0$ to $t = t_f$. The trajectories $x(t)$ and $\hat{x}(t)$ track the lower and upper bounds of the interval over-approximation respectively, so that $\Omega(\mathcal{X}_{in}) = [x(t_f),\, \hat{x}(t_f)] \supseteq \mathcal{R}(\mathcal{X}_{in})$.

The key feature of this reachability step for the verifier described in Section 4 lies in its low computational cost, as a single CTMM call requires one Jacobian bounding, one L_x assembly, and one $2n$-dimensional ODE integration.

4 TNODEV Verifier Architecture

TNODEV is a formal verifier for neural ODE systems, with a modular architecture, and it can be easily extended to encompass other reachability analysis methods, model classes, and specification types.

Definition 3 (Safety specification)**.** *A safety specification is a pair $(\mathcal{X}_{in}, \mathcal{P})$, where $\mathcal{X}_{in} \subseteq \mathbb{R}^n$ is an initial input set and $\mathcal{P} : \mathbb{R}^n \to \{\mathsf{true}, \mathsf{false}\}$ is a property on the state at time t_f. The specification is satisfied if $\mathcal{P}\big(\Phi(t_f, u)\big) = \mathsf{true}$ for every $u \in \mathcal{X}_{in}$. TNODEV currently supports two types of properties, both reducing to safe-set inclusion $x \in \mathcal{X}_s$ for an interval-defined or implicitly-defined safe set $\mathcal{X}_s$:*

- Interval safe set*: $\mathcal{P}(x) \;\Leftrightarrow\; x \in \mathcal{X}_s$, where $\mathcal{X}_s = [\underline{s}, \overline{s}] \subseteq \mathbb{R}^n$ is an axis-aligned interval given explicitly as part of the specification.*
- Classification robustness safe set*: $\mathcal{P}(x) \;\Leftrightarrow\; \arg\max_j x_j = y^\star$ for a fixed target label $y^\star$, equivalently $x \in \mathcal{X}_s$ with $\mathcal{X}_s = \bigcap_{j \neq y^\star} \{x \in \mathbb{R}^n : x_{y^\star} \geq x_j\}$ the intersection of half-spaces induced by $y^\star$.*

Any property $\mathcal{P}$ that admits a sound check on the reachable set over-approximation $\Omega(\mathcal{X})$ can be plugged into the verification loop without modifying the rest of the pipeline. Extending the interface to other property types (e.g., reach-avoid specifications, or polyhedral sets defined by linear half-space constraints rather than axis-aligned intervals) requires only implementing the appropriate check on $\Omega(\mathcal{X})$. On the model side, TNODEV supports three classes of neural ODE models: a pure neural ODE, a general neural ODE that combines an ODE block with discrete neural network layers (Manzanas Lopez et al., 2022), and a hybrid neural ODE control system in which the ODE plant is wrapped by a neural network controller.

Given a specification, the verifier returns one of three verdicts: `SAFE` (the specification is formally verified), `FALSIFIED` (a concrete counter-example was found), or `UNKNOWN` (no decision was reached within the maximum number of refinement iterations K or the wall-clock timeout $T_{\max}$). The soundness of the methods used within TNODEV ensures that the `SAFE` and `FALSIFIED` outcomes respectively guarantee the satisfaction and falsification of the specification by the considered neural ODE. However, without infinite computational power, TNODEV cannot be considered as complete, which is why the algorithm may still return `UNKNOWN` when no other outcome was reached within the allocated computational limits.

The architecture comprises four components: a falsification module, a specification interface, an iterative verification loop that combines reachability and refinement, and a parallel scheduler. Their interaction is summarized in Algorithm 1 and illustrated in Figure 1.

4.1 Falsification

Before any reachability call, TNODEV runs a lightweight falsification pass on the entire initial set $\mathcal{X}_{in}$. A set of samples (corners, centroid, and uniform random points) is drawn from $\mathcal{X}_{in}$ and each sample is integrated forward to time t_f using the neural ODE dynamics. If any sample trajectory ends in a state where $\mathcal{P}$ is violated, the verifier terminates with a `FALSIFIED` verdict and returns the violating trajectory as a concrete counter-example, as illustrated in Figure 2(a). Otherwise, the verifier proceeds to the verification loop. Placing falsification before reachability avoids spending refinement iterations on input sets that contain obvious violations, and a `FALSIFIED` verdict is conclusive since it exhibits an explicit counter-example.

4.2 Specification Interface

For any cell $\mathcal{X} \subseteq \mathcal{X}_{in}$ in the refinement queue, the reachability block computes an over-approximation $\Omega(\mathcal{X}) \supseteq \mathcal{R}(\mathcal{X})$. The specification interface checks whether $\Omega(\mathcal{X})$ satisfies the property $\mathcal{P}$, with the implementation depending on the property type.

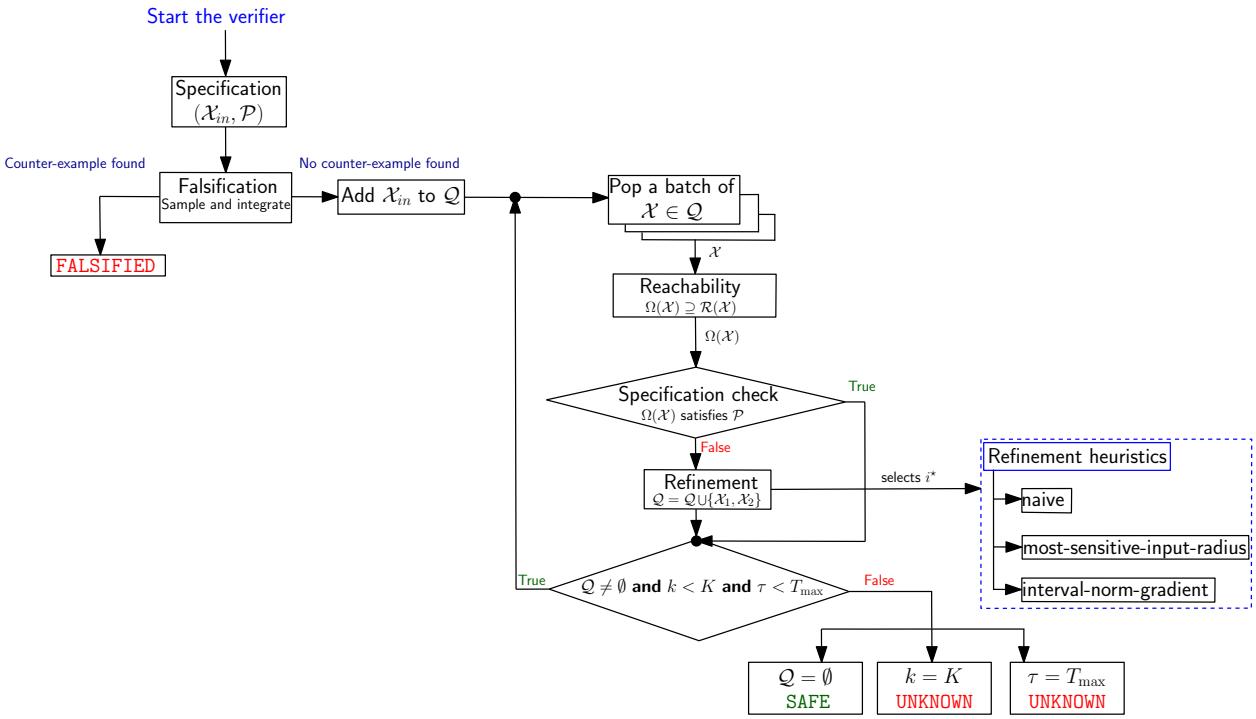


Figure 1: TNODEV verifier architecture. The verifier starts with a falsification check on the full initial set $\mathcal{X}_{in}$. If a counter-example is found, it returns `FALSIFIED`; otherwise it enters the verification and refinement loop, where the reachability block computes the over-approximation $\Omega(\mathcal{X})$ for each cell, the specification check verifies whether $\Omega(\mathcal{X})$ satisfies the property $\mathcal{P}$, and the refinement block splits inconclusive cells using one among three heuristics. The loop terminates with `SAFE` when the queue $\mathcal{Q}$ is empty, or `UNKNOWN` if the refinement iteration limit K or the wall-clock timeout $T_{\max}$ is reached.

For an interval safe-set inclusion $\mathcal{X}_s = [\underline{s}, \overline{s}]$, the over-approximation is the interval $\Omega(\mathcal{X}) = [\underline{x}, \overline{x}]$ returned by the reachability block, and the check $\Omega(\mathcal{X}) \subseteq \mathcal{X}_s \iff \underline{x} \geq \underline{s}$ and $\overline{x} \leq \overline{s}$ reduces to component-wise interval comparisons.

For classification robustness, the property is encoded as a set of half-spaces $H_j = \{x \in \mathbb{R}^n : x_{y^\star} - x_j \leq 0\}$ for each $j \neq y^\star$ (each H_j is the region where competing class j ties or beats the target $y^\star$), and the cell is verified if $\Omega(\mathcal{X}) \cap H_j = \emptyset$ for all j.

If the check passes, the cell is verified as `SAFE`. Otherwise, the verdict on this cell is inconclusive, as the violation may correspond to either an unsafe trajectory, or a loose over-approximation. The cell is then forwarded to the refinement step.

4.3 Verification & Refinement Loop

If no counter-example is found during the falsification phase, the verifier enters the verification and refinement loop. The loop maintains a queue of input cells $\mathcal{Q}$ initialized to $\{\mathcal{X}_{in}\}$ and iterates until all cells have been verified, or the refinement iteration limit K is exhausted, or the wall-clock timeout $T_{\max}$ is reached. At each iteration, a batch of cells $\mathcal{B}$ is popped from $\mathcal{Q}$. The batch size b is a fixed parameter, chosen to match the number of available parallel workers so that each iteration dispatches a full round of independent cells to the parallelization pool (Section 4.4). When fewer than b cells remain, the whole queue forms the batch, i.e., $|\mathcal{B}| = \min(|\mathcal{Q}|, b)$. Each cell $\mathcal{X}$ in the batch is processed by a reachability call to obtain $\Omega(\mathcal{X})$, followed by the specification check of Section 4.2. Cells that pass the check are marked `SAFE`, and cells that do not pass are divided into two sub-cells $\mathcal{X}_1, \mathcal{X}_2$ by the refinement step and added to $\mathcal{Q}$. The loop terminates with

Algorithm 1 TNODEV verification loop

Input: neural ODE f, specification $(\mathcal{X}_{in}, \mathcal{P})$, final time t_f, maximum number of refinement iterations K, wall-clock timeout $T_{\max}$, batch size b

▷ **Phase 1: Falsification**

Sample trajectories from $\mathcal{X}_{in}$ (corners, centroid, random)
Integrate each sample forward to t_f
if any sample violates $\mathcal{P}$ **then**
    return `FALSIFIED`, violating trajectory
end if

▷ **Phase 2: Verification and Refinement**

$\mathcal{Q} \leftarrow \{\mathcal{X}_{in}\}$ ▷ queue of input cells
$k \leftarrow 0$ and start timer τ
while $\mathcal{Q} \neq \emptyset$ **and** $k < K$ **and** $\tau < T_{\max}$ **do**
    pop a batch $\mathcal{B} \subseteq \mathcal{Q}$ with $|\mathcal{B}| = \min(|\mathcal{Q}|, b)$
    for all $\mathcal{X} \in \mathcal{B}$ **in parallel do**
        $\Omega(\mathcal{X}) \leftarrow \textsc{Reach}(f, \mathcal{X}, t_f)$
        if $\Omega(\mathcal{X})$ satisfies $\mathcal{P}$ **then**
            mark $\mathcal{X}$ as `SAFE`
        else
            $\mathcal{X}_1, \mathcal{X}_2 \leftarrow \textsc{Refine}(\mathcal{X})$
            push $\mathcal{X}_1, \mathcal{X}_2$ onto $\mathcal{Q}$
        end if
    end for
    $k \leftarrow k + 1$
end while
if $\mathcal{Q} = \emptyset$ **then**
    return `SAFE`
else
    return `UNKNOWN`
end if

a `SAFE` verdict when $\mathcal{Q}$ is empty (i.e., all cells are verified), or with an `UNKNOWN` verdict if K or $T_{\max}$ are reached while some cells are still in the queue.

Refinement. When a cell $\mathcal{X}$ with center x and half width r cannot be verified, the refinement block divides it along a single input dimension $i^\star$ chosen by a heuristic, producing two sub-cells of equal half width along $i^\star$. We implement and compare three heuristics, illustrated on the Spiral 2D benchmark in Figure 2(b)–(d).

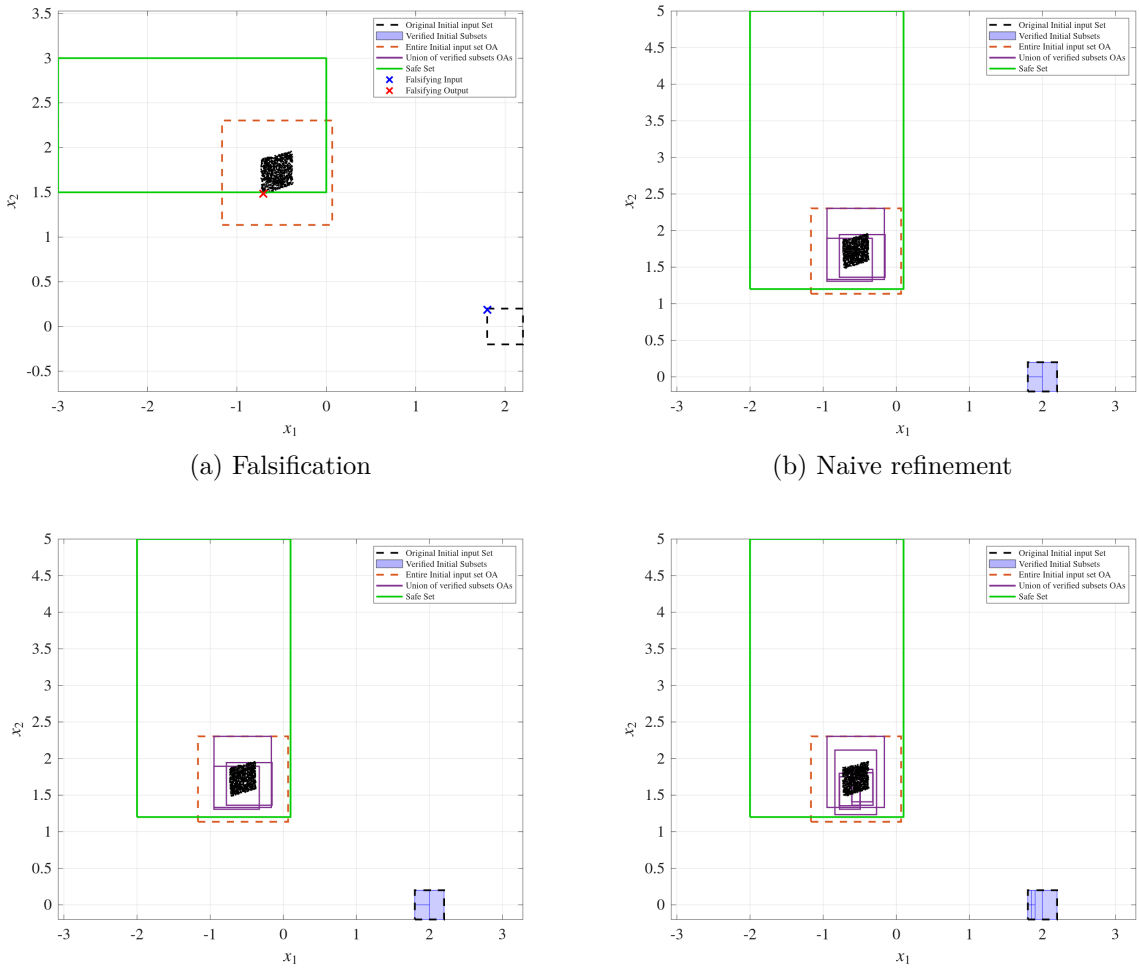
Naive. Split the dimension with the largest interval radius: $i^\star = \arg\max_i r_i$. This is the standard baseline used by most verification tools.

Most-sensitive-input-radius (MSIR). For each input dimension i, compute the product of the cell's radius along i and a worst case Jacobian magnitude in column i:

$$s_i \;=\; r_i \cdot \max_k \max\big(|\underline{J}_{ki}|,\ |\overline{J}_{ki}|\big),$$

where $[\underline{J}, \overline{J}]$ are interval Jacobian bounds of f over the cell. The split dimension is $i^\star = \arg\max_i s_i$. The Jacobian bounds are obtained through interval propagation on the symbolic Jacobian of the network, so MSIR requires no additional reachability calls.

Interval-norm-gradient (ING). Estimate the partial derivatives of the over-approximation width $w(\mathcal{X}) = \sum_k (\overline{x}_k - \underline{x}_k)$ with respect to the cell's center by finite differences. For each input dimension j, perturb the center by a small offset δ along j, keep the radius fixed, re-run the reachability call to obtain $w(\mathcal{X}_{j,\delta})$, and



(a) Falsification (b) Naive refinement

(c) MSIR refinement (d) ING refinement

Figure 2: TNODEV verification results on the Spiral 2D benchmark. In all sub figures, the dashed black rectangle is the original initial set $\mathcal{X}_{in}$, the dashed red rectangle is the over-approximation $\Omega(\mathcal{X}_{in})$ from a single reachability call on the unsplit initial set, the green rectangle is the safe set $\mathcal{X}_s$, the violet rectangles represent the verified subsets, and the black dots are sampled successor trajectories.

let $g_j \approx \partial w / \partial x_j$ denote the resulting finite-difference estimate:

$$g_j \;\approx\; \frac{w(\mathcal{X}_{j,\delta}) - w(\mathcal{X})}{\delta}.$$

The split dimension is $i^\star = \arg\max_j |g_j|$. ING requires n additional reachability calls per refinement decision, i.e., one per input dimension.

The three heuristics span a cost spectrum: naive uses no extra computation, MSIR adds only Jacobian-bounding cost (negligible relatively to a reachability call), and ING adds n reachability calls per refinement decision. In practice, MSIR offers the best overall trade-off, as it reduces the number of refinement iterations through informed splitting decisions without incurring the per-decision overhead of ING (see Table 2 and the per-benchmark analysis in Appendix C where MSIR achieves the lowest verification time on the FPA benchmark and the largest iteration reduction on the higher-dimensional Cartpole benchmark). This positions MSIR as the optimal default heuristic for TNODEV.

4.4 Parallelization

Because each cell in the refinement queue $\mathcal{Q}$ can be processed independently, the inner loop of Algorithm 1 (lines 10–18) is run in parallel. At each iteration, TNODEV pops a batch of $|\mathcal{B}| = \min(|\mathcal{Q}|, b)$ cells and distributes them across CPU workers, so that each worker runs the reachability and specification check on its assigned cell and either marks it `SAFE` or produces the two refined sub-cells. The refined sub-cells are pushed back onto $\mathcal{Q}$ at the end of the iteration, and the next batch is popped only once the whole batch completes. Workers within a batch operate on disjoint cells and need no synchronization among themselves.

5 Experiments and Results

5.1 Reachability Analysis Comparison

To the best of our knowledge, TNODEV is the first formal verifier for neural ODE to combine reachability, falsification, and iterative refinement in a single pipeline. As a consequence, no other existing tool exposes an equivalent end-to-end workflow to compare against, and the range of head-to-head verification comparisons is consequently limited. To compensate for that, we start the experimental evaluation with a direct comparison of TNODEV core reachability component against the two closest neural ODE reachability tools in the literature: NNV 2.0 (Lopez et al., 2023) and CORA (Althoff, 2015). The comparison is carried out on the Spiral nonlinear and FPA benchmarks (defined in Appendix A), where all three tools compute a single reachability call. We measure the wall-clock time of one reachability call, and the size of the resulting over-approximation reported as a single dimension-comparable metric, the geometric mean width $\mu(\Omega) = \sqrt[n]{\mathrm{Vol}(\Omega)}$, defined and detailed in Appendix B.1.

The results highlight a consistent qualitative trade-off across the three tools. On the Spiral nonlinear benchmark, TNODEV is 14× faster than NNV 2.0 and 83× faster than CORA, as reported in Table 5. On FPA, TNODEV is approximately 7× faster than NNV 2.0 and 39× faster than CORA, as reported in Table 6. However, this gain in terms of computation time comes at the cost of the tightness of the over-approximation. Namely, on Spiral nonlinear, NNV 2.0 Star set and CORA zonotope produce over-approximations respectively 4× and 4.3× tighter than TNODEV interval over-approximation. On FPA, the results are more mixed, as TNODEV interval over-approximation is approximately 1.3× tighter than NNV 2.0 Star set, while CORA's zonotope remains the tightest at 1.4× tighter than TNODEV.

This speed-tightness trade-off motivates the verifier architecture of Section 4, as instead of relying on a single reachability call to be tight, TNODEV wraps a fast reachability backend in the verification and refinement loop that splits the input set and re-compute the reachability on each sub-cell until the over-approximation becomes tight enough for the property to be certified, or the refinement iteration limit is reached. The remainder of the section evaluates this loop in two regimes. Section 5.2 exercises the loop on six safe-set inclusion benchmarks across pure and hybrid neural ODE model classes. Section 5.3 then evaluates the scalability of TNODEV on the high-dimensional regime of GNODE classifiers, and provides a direct verification comparison against NNV 2.0, which can run on these benchmarks but offers only a single reachability call without iterative refinement.

5.2 Interval safe set inclusion benchmarks

We evaluate TNODEV on four pure neural ODE benchmarks (Spiral 2D linear and nonlinear, FPA 5D, Cartpole 12D) and two hybrid neural ODE control system benchmarks (ACC 8D linear and nonlinear plant variants) as described in Appendix A. Each benchmark is run with the three refinement heuristics introduced in Section 4.3. We set $K = 5000$ as the maximum number of refinement iterations and $T_{\max} = 120$ minutes as the wall-clock timeout, for all benchmarks. Table 2 reports the results for each benchmark. Out of the eighteen benchmark-heuristic combinations, fourteen terminate with a `SAFE` verdict. The other four return `UNKNOWN` due to the ING heuristic either exhausting the refinement limit on Spiral linear and Cartpole, or reaching the wall-clock timeout on both ACC variants. The fourteen `SAFE` verdicts demonstrate that TNODEV verifies the specified safety properties across input dimensions ranging from 2 to 12. The four

Table 2: TNODEV verification results across all benchmarks. Each benchmark is run with three refinement heuristics. Iterations and verified subsets refer to the verification and refinement loop (Algorithm 1). Time is the total verification time.

Benchmark	Dim.	Heuristic	Time	Iter.	Verdict	Verified Subsets
Spiral linear	2	Naive	6.79 s	31	SAFE	16
		MSIR	6.50 s	29	SAFE	15
		ING	13.16 min	5000	UNKNOWN	242
Spiral nonlinear	2	Naive	15.60 s	5	SAFE	3
		MSIR	10.78 s	3	SAFE	2
		ING	46.96 s	11	SAFE	6
FPA	5	Naive	53.78 s	17	SAFE	9
		MSIR	34.50 s	9	SAFE	5
		ING	53.78 s	3	SAFE	2
Cartpole	12	Naive	2.73 min	2037	SAFE	1019
		MSIR	56.43 s	63	SAFE	32
		ING	54.03 min	5000	UNKNOWN	0
ACC linear	8	Naive	4.78 min	15	SAFE	8
		MSIR	2.72 min	7	SAFE	4
		ING	120 min	1087	UNKNOWN	0
ACC nonlinear	8	Naive	20.03 min	41	SAFE	21
		MSIR	29.61 min	37	SAFE	19
		ING	120 min	607	UNKNOWN	0

`UNKNOWN` verdicts are further discussed in Appendix C. All of the specifications for the benchmarks are given in Appendices B.2–B.6

Pure neural ODE benchmarks. On the small scale benchmarks (Spiral 2D and FPA 5D), the verification and refinement loop runs for multiple refinement iterations, ranging from 3 to 5000, depending on the heuristic and the property $\mathcal{P}$ being verified. The number of verified subsets and the verification time vary between heuristics as illustrated in Figures 3–7 of Appendix B, and we discuss this trade-off in Appendix C. On the Spiral linear variant, the ING heuristic reaches the refinement limit without converging, producing a total of 242 verified subsets but ultimately returning an `UNKNOWN` verdict, due to repeated subdivision without sufficient progress. On the 12D Cartpole benchmark, the verification and refinement loop capabilities are illustrated for the naive heuristic resulting in 2037 refinement iterations and 1019 verified subsets, and the MSIR heuristic reduces this amount to 63 refinement iterations and 32 verified subsets. The ING heuristic also fails to converge on this benchmark, reaching the refinement limit without verifying a single subset and returns `UNKNOWN`, reflecting the compounding cost of its n extra reachability calls per refinement decision on higher dimensional input space.

Hybrid neural ODE control system benchmark. On the ACC 8D benchmark, both the linear and nonlinear plant variants are verified `SAFE` using the naive and MSIR heuristics, exercising the refinement loop in the hybrid closed-loop setting where a neural network controller wraps the neural ODE plant. On the linear variant, the verification achieved based on the MSIR heuristic takes 2.72 min over 7 iterations against 4.78 min over 15 iterations for the naive heuristic. On the nonlinear variant, the verification based on the naive heuristic takes 20.03 min over 41 iterations while it takes 29.61 min over 37 iterations for the MSIR heuristic. It is worth noting that the higher cost of the nonlinear variant over the linear one reflects the higher per-step reachability cost of the tanh-activated plant. The ING heuristic fails to converge on both

variants, reaching the wall-clock timeout without verifying any subset and returning `UNKNOWN`, consistent with the compounding cost of its n extra reachability calls per refinement decision observed on the other higher-dimensional benchmarks (see the per-benchmark analysis in Appendix C).

5.3 Classification neural ODE benchmark

We additionally evaluate TNODEV on a classification robustness benchmark built around a general neural ODE (GNODE) classifier (Manzanas Lopez et al., 2022) for the MNIST handwritten-digit data set. Full architectural details, the hybrid reachability pipeline, and the bound-extraction modes used at the ODE-block boundary are given in Appendix A.3. The verification problem on this benchmark is to certify L_∞ adversarial robustness around individual test images, which we formalize as a specific instance of Definition 3.

Definition 4 (L_∞ adversarial robustness)**.** *Let f be a neural ODE classifier whose final state at time t_f is the logit vector $\Phi(t_f, u) \in \mathbb{R}^n$ over n classes, and let $(x_0, y^\star)$ be a clean input label pair for which f is correct, i.e.,* $\arg\max_j \Phi_j(t_f, x_0) = y^\star$. *We say f is robust at $(x_0, y^\star)$ if the safety specification $(\mathcal{X}_{in}, \mathcal{P})$ is satisfied, where*

$$\mathcal{X}_{in} = B_\infty(x_0, \epsilon) = \{u \in \mathbb{R}^n : \|u - x_0\|_\infty \leq \epsilon\}$$

is the L_∞ ball of radius ϵ around x_0, and $\mathcal{P}$ is the classification robustness property with target label $y^\star$:

$$\mathcal{P}(x) \;\Leftrightarrow\; \arg\max_j x_j = y^\star.$$

A classifier verified ϵ-robust at $(x_0, y^\star)$ is guaranteed to predict $y^\star$ on all adversarially perturbed inputs $u \in B_\infty(x_0, \epsilon)$, formally certifying it against any L_∞-bounded attack of magnitude up to ϵ.

We compare TNODEV with NNV 2.0 on the two convolutional GNODE classifiers CNODE_S and CNODE_M of Manzanas Lopez et al. (2022). TNODEV current version targets the verification of neural ODE, but does not yet provide reachability for the neural network layers that surround the ODE block in a GNODE architecture. We therefore construct a hybrid pipeline in Appendix A.3.2 in which NNV 2.0 handles the pre-ODE and post-ODE layers, while TNODEV handles the linear ODE block via a closed-form matrix-exponential propagator using interval sets. We state its construction formally and prove its soundness in Appendix A.3.2, and discuss the empirical justification for the design choice there as well. We report TNODEV under two bound-extraction modes (LP-tight and fast) to convert into an interval the sets coming into the ODE block, and this is discussed in more detail in Appendix A.3.3. We evaluate on the first 50 MNIST test images at $\epsilon = 0.5/255$, matching the sample size used in Manzanas Lopez et al. (2022), and report per-image and total computational times for both tools. Neither of the two tools uses iterative refinement in this comparison, as NNV 2.0 does not currently implement input-set refinement, and it manages to obtain its verification verdicts without it. Therefore, to keep the comparison meaningful, we use TNODEV here as well without refinements, and we leave adding the refinement for this benchmark, applied to the global GNODE input set rather than to the ODE block input for future work.

Table 3 shows that NNV 2.0 verifies nearly all images on both networks (49/50 on CNODE_S, and 50/50 on CNODE_M). TNODEV verifies 29/50 (58%) on CNODE_S and 30/50 (60%) on CNODE_M, with the LP-tight and fast bound-extraction modes returning identical verdict counts on this benchmark, indicating that the interval over-approximation through the linear ODE is the binding constraint, rather than the predicate-constraint information discarded at the ODE-block boundary (Appendix A.3.3).

Where NNV 2.0 wins. The verification gap originates entirely from the linear ODE block, or more precisely from the fact that the true reachable set of the linear ODE block from the input interval is in general a polytope rather than an axis-aligned box. NNV 2.0 represents this reachable set exactly using a predicate-constrained Star set, while TNODEV represents it by the tightest axis-aligned box that contains it, which is a sound enclosure but discards the polytopal structure whenever the reachable set is not itself axis-aligned. The empirical widening observed through the linear ODE is roughly 7× in aggregate width on CNODE_S and 8× on CNODE_M at $\epsilon = 0.5/255$. The MNIST benchmark consists of a single linear ODE block, a setting in which Star set propagation is exact and interval based propagation is not, which is why TNODEV is at a structural disadvantage here. The verification rate shortfall is therefore the tightness side

Table 3: Robustness verification of MNIST GNODE classifiers under a full-image L_∞ perturbation of $\epsilon = 0.5/255$. n denotes the dimension of the linear ODE state in each network (equal to the flattened pre-ODE feature dimension, refer to Table 4).

Network	**Method**	**Robust/50**	**Time/img. (s)**	**Total (s)**
CNODE_S ($n = 676$)	NNV 2.0	49	10.83	541.6
	TNODEV (LP-tight)	29	22.97	1148.6
	TNODEV (fast)	29	2.20	109.9
CNODE_M ($n = 1690$)	NNV 2.0	50	166.4	8321.9
	TNODEV (LP-tight)	30	35.87	1793.5
	TNODEV (fast)	30	0.83	41.4

of the same speed-tightness trade-off observed in the reachability comparison of Appendix B.1, as TNODEV is fast but with wider over-approximations, and on a benchmark dominated by a single exactly-representable linear block, that wideness translates directly into fewer verified images (Appendix A.3.4 expands on this comparison). The interval over-approximation through the linear ODE also imposes an empirical limitation on the perturbation magnitudes for which TNODEV produces useful verdicts on this benchmark, Appendix A.3.5 discusses this applicability scope in more detail.

Where TNODEV wins. TNODEV is between 5× and 200× faster than NNV 2.0 on a per-image basis, with the gap widening sharply as the ODE state dimension increases. On CNODE_M the closed-form interval propagation requires only 0.83 s per image against NNV 2.0's 166.4 s, which corresponds to a 200× speed up. On the smaller CNODE_S, NNV 2.0's per-image cost drops to 10.83 s, slightly faster than TNODEV LP-tight mode at 22.97 s because the LP-based bound extraction at the ODE-input boundary becomes the dominant cost on small networks, but the fast mode bypasses this bottleneck and recovers a 5× speedup at 2.20 s.

6 Conclusion

We present TNODEV, a formal verifier for neural ODEs. To the best of our knowledge, TNODEV is the first verifier for neural ODE to integrate refinement-based precision recovery into the workflow, addressing a structural gap in the current neural ODE verification literature where existing tools provide only a single reachability call without any iterative refinement.

The experimental results in Section 5 feature both the strengths and the limitations of TNODEV current version. On the reachability comparison against NNV 2.0 and CORA, TNODEV is one to two orders of magnitude faster while producing a bit looser over-approximations, illustrating the speed and tightness trade-off intrinsic to interval-based propagation. On the safe-set inclusion benchmarks across pure neural ODE and hybrid neural ODE control system model classes, TNODEV verifies benchmarks ranging from 2 to 12 input dimensions, with the MSIR heuristic emerging as the best general-purpose default among the three studied heuristics. On the high dimensional MNIST classification benchmark, the verification rate of TNODEV is lower than NNV 2.0 due to the structural disadvantage of interval propagation on linear ODE blocks, but the per-image verification time is between 5× and 200× lower.

Some of the limitations of the current TNODEV version identified in the previous section suggest possible directions for future improvements of the toolbox. Firstly, for neural ODE described by a linear ODE, it would be interesting to explore the inclusion of alternative reachability methods relying on richer set representations (such as zonotopes or polynomial zonotopes) to more precisely approximate the reachable set than what the current interval approach can achieve. Next, we want to design more robust gradient-style refinement heuristics that retain ING's positive splitting behavior while reducing its higher computational cost. Finally, we would like to include in TNODEV an implementation of reachability methods of discrete neural network layers, to avoid depending on external tools in the context of GNODE systems.

References

Matthias Althoff. An introduction to CORA 2015. In *Workshop on Applied Verification for Continuous and Hybrid Systems*, pp. 120–151, 2015.

Panos J. Antsaklis and Anthony N. Michel. *A Linear Systems Primer*. Birkhäuser, 2007.

Randall D. Beer. On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):469–509, 1995.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42): 1–39, 2020.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Samuel Coogan. Mixed monotonicity for reachability and safety in dynamical systems. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 5074–5085. IEEE, 2020.

Martin E. Dyer and Alan M. Frieze. On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing*, 17(5):967–974, 1988.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Sophie Gruenbacher, Jacek Cyranka, Mathias Lechner, Md Ariful Islam, Scott A Smolka, and Radu Grosu. Lagrangian reachtubes: The next generation. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 1556–1563. IEEE, 2020.

Sophie Gruenbacher, Mathias Lechner, Ramin Hasani, Daniela Rus, Thomas A. Henzinger, Scott Smolka, and Radu Grosu. GoTube: Scalable stochastic verification of continuous-depth models. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence*, 2022.

Sophie Grunbacher, Ramin Hasani, Mathias Lechner, Jacek Cyranka, Scott A. Smolka, and Radu Grosu. On the verification of neural ODEs with stochastic guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 11525–11535, 2021.

Leonid Genrikhovich Khachiyan. The problem of calculating the volume of a polyhedron is enumerably hard. *Russian Mathematical Surveys*, 44(3):199–200, 1989.

Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10):642–652, 2020.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Zhen Liang, Dejin Ren, Bai Xue, Ji Wang, Wenjing Yang, and Wanwei Liu. Verifying safety of neural networks from topological perspectives. *Science of Computer Programming*, 238:103160, 2024.

Changliu Liu, Tomer Arnon, Chris Lazarus, Christopher Strong, Clark Barrett, and Mykel J Kochenderfer. Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization*, 4(3-4):244–404, 2021.

Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T. Johnson. NNV 2.0: The neural network verification tool. In *International Conference on Computer Aided Verification (CAV)*, 2023.

Diego Manzanas Lopez, Patrick Musau, Nathaniel P. Hamilton, and Taylor T. Johnson. Reachability analysis of a general class of neural ordinary differential equations. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, pp. 258–277. Springer, 2022.

Pierre-Jean Meyer, Alex Devonport, and Murat Arcak. TIRA: Toolbox for interval reachability analysis. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

Pierre-Jean Meyer, Alex Devonport, and Murat Arcak. *Interval Reachability Analysis: Bounding Trajectories of Uncertain Systems with Boxes for Control and Verification*. Springer Nature, 2021.

Intae Moon, Stefan Groha, and Alexander Gusev. SurvLatent ODE: A neural ODE based time-to-event model with competing risks for longitudinal data improves cancer-associated venous thromboembolism (VTE) prediction. In *Machine Learning for Healthcare Conference*, pp. 800–827. PMLR, 2022.

Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*. SIAM, 2009.

P Musau and TT Johnson. Continuous-time recurrent neural networks (ctrnns) (benchmark proposal). In *5th Applied Verification for Continuous and Hybrid Systems Workshop (ARCH), Oxford, UK*, 2018. URL `https://doi.org/10.29007/6czp`.

Zhaozhi Qian, William Zame, Lucas Fleuren, Paul Elbers, and Mihaela van der Schaar. Integrating expert ODEs into neural ODEs: pharmacology and disease progression. *Advances in Neural Information Processing Systems*, 34:11364–11383, 2021.

R. Rico-Martínez, I. G. Kevrekidis, and K. Krischer. Continuous-time modeling of nonlinear systems: A neural network-based approach. *Chemical Engineering Communications*, 1992.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Abdelrahman Sayed Sayed, Pierre-Jean Meyer, and Mohamed Ghazel. Bridging neural ODE and ResNet: A formal error bound for safety verification. In *International Symposium on AI Verification*, pp. 97–114. Springer, 2025a.

Abdelrahman Sayed Sayed, Pierre-Jean Meyer, and Mohamed Ghazel. Mixed monotonicity reachability analysis of neural ODE: A trade-off between tightness and efficiency. *arXiv preprint arXiv:2510.17859*, 2025b.

Hoang-Dung Tran, Diego Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-based reachability analysis of deep neural networks. In *International Symposium on Formal Methods (FM)*, pp. 670–686. Springer, 2019.

Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification (CAV)*, pp. 3–17. Springer, 2020.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in neural information processing systems*, 34:29909–29921, 2021.

Tianhao Wei, Hanjiang Hu, Luca Marzari, Kai Sheng Yun, Peizhi Niu, and Changliu Liu. ModelVerification.jl: A comprehensive toolbox for formally verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, pp. 395–408. Springer, 2025.

Wei Xiao, Tsun-Hsuan Wang, Ramin Hasani, Mathias Lechner, Yutong Ban, Chuang Gan, and Daniela Rus. On the forward invariance of neural ODEs. In *International conference on machine learning*, pp. 38100–38124. PMLR, 2023.

Appendix

Experimental Settings: All the experiments[1] herein are run on MATLAB R2026a with the Continuous Reachability Analyzer (CORA) version 2025.1.1, the Toolbox for Interval Reachability Analysis (TIRA) version 2, and the Neural Network Verification Software Tool (NNV 2.0) on an AMD Ryzen™ 9 5950X CPU (16 cores, 32 threads) with 96 GB of RAM.

A System description

This appendix describes the neural ODE benchmarks used in the experiments, organized by the three model classes introduced in Section 4. Spiral and FPA are pure neural ODE in which the entire right-hand side f of equation 1 is parameterized by a neural network. Cartpole is also presented as a single ODE, but its right-hand side combines a neural ODE controller with cartpole physics equations rather than being entirely a neural ODE. ACC is a hybrid neural ODE control system in which a neural ODE plant is controlled by a separate neural network controller in a discrete-time closed loop setting. MNIST is a general neural ODE (GNODE) in which an ODE block is embedded in a feed-forward classifier between discrete neural network layers.

A.1 Pure neural ODE benchmarks

A.1.1 Spiral

The spiral system is a 2-dimensional neural ODE (Chen et al., 2018) whose trajectories form a spiral in the state space. We consider two variants from the literature, a *nonlinear* variant from Chen et al. (2018) and a *linear* variant from Manzanas Lopez et al. (2022). The two variants share the same architecture of a 2-layer feed-forward network with a hidden layer of 10 neurons, in which the input is in $\mathbb{R}^2$, the hidden layer is in $\mathbb{R}^{10}$, and the output is again in $\mathbb{R}^2$. The two variants differ only in the activation function applied to the hidden layer.

Nonlinear

In the nonlinear variant, the activation is tanh, and the neural ODE has the following dynamics

$$\dot{x} = f(x) = W_2 \tanh(W_1 x + b_1) + b_2,$$

where $x \in \mathbb{R}^2$, $W_1 \in \mathbb{R}^{10\times 2}$, $b_1 \in \mathbb{R}^{10}$, $W_2 \in \mathbb{R}^{2\times 10}$, $b_2 \in \mathbb{R}^2$, and $\tanh(\cdot)$ is applied element-wise to the hidden vector $W_1 x + b_1 \in \mathbb{R}^{10}$. The exact values of the weight matrices and bias vectors are defined within the MATLAB function *System_description.m*.

Linear

In the linear variant, there is no activation function, and the neural ODE has the following dynamics

$$\dot{x} = f(x) = W_2(W_1 x + b_1) + b_2,$$

with W_1, b_1, W_2, b_2 of the same shapes as in the nonlinear variant above. Since the hidden layer applies no activation, the dynamics reduce to a 2-dimensional linear ODE $\dot{x} = Ax + b$ with $A = W_2 W_1 \in \mathbb{R}^{2\times 2}$ and $b = W_2 b_1 + b_2 \in \mathbb{R}^2$. The exact values of the weight matrices and bias vectors are defined within the MATLAB function *System_description.m*.

A.1.2 FPA

The Fixed-Point Attractor (FPA) system is a 5-dimensional neural ODE that admits a fixed-point equilibrium under appropriate conditions, originating from a continuous-time recurrent neural network model in Beer

[1]Code available in the following repository: `https://doi.org/10.5281/zenodo.20546365`

(1995) and adopted as a verification benchmark by Musau & Johnson (2018). We consider here the same 5-dimensional formulation, with the following neural ODE dynamics

$$\dot{x} = f(x) = \tau x + W \tanh(x),$$

where $x \in \mathbb{R}^5$, $\tau = -10^{-6}$ is a small linear-leak coefficient, and $W \in \mathbb{R}^{5\times 5}$ is a composite recurrent weight matrix

$$W = \begin{pmatrix} 0_{2\times 2} & A \\ 0_{3\times 2} & BA \end{pmatrix},$$

The matrices A and B together fully specify W, and the corresponding values are loaded directly from the MATLAB function *System_description.m*.

A.1.3 Cartpole

The Cartpole benchmark we use was introduced as a neural ODE reachability problem in Gruenbacher et al. (2020; 2022). The cart is moving along a frictionless track and it must keep an attached pole upright by applying a horizontal force, with the controller realized as a continuous-time recurrent neural network (CT-RNN). The physical state collects the cart position x_{cart}, cart velocity v_{cart}, pole angle θ, and pole angular velocity ω, and the controller maintains 8 recurrent hidden states $h_1, \ldots, h_8$ whose dynamics are integrated jointly with the physics, giving an overall 12-dimensional neural ODE with state vector $x = (x_{\text{cart}}, v_{\text{cart}}, \theta, \omega, h_1, \ldots, h_8)^\top \in \mathbb{R}^{12}$.

The dynamics consists of two coupled blocks. The first is the standard cartpole physics, in which the four physical components of $\dot{x}$ are given by

$$\dot{x}_{\text{cart}} = v_{\text{cart}}, \quad \dot{v}_{\text{cart}} = a_{\text{cart}}, \quad \dot{\theta} = \omega, \quad \dot{\omega} = a_{\text{pole}},$$

with the cart and pole accelerations a_{cart} and a_{pole} obtained from the standard cartpole equations of motion as functions of θ, ω, and the applied force $F = F_{\text{mag}} \cdot u$, where $F_{\text{mag}} = 10$ N is the actuator magnitude and $u \in [-1, 1]$ is the controller's scalar action defined below.

The second block is the CT-RNN controller, in which the hidden-state vector $h = (h_1, \ldots, h_8)^\top$ evolves according to

$$\dot{h} = -h + \tanh\big(W^{\text{rec}} h + W^{\text{in}} p + b\big),$$

where $p = (x_{\text{cart}}, v_{\text{cart}}, \theta, \omega)^\top$ is the physical sub-state fed to the controller, $W^{\text{rec}} \in \mathbb{R}^{8\times 8}$ is the recurrent weight matrix, $W^{\text{in}} \in \mathbb{R}^{8\times 4}$ is the input weight matrix, $b \in \mathbb{R}^8$ is the bias vector, and $\tanh(\cdot)$ is applied element-wise. The self-decay term $-h$ causes the hidden state to relax exponentially toward zero in the absence of input, while the tanh term provides a bounded driving signal that mixes the recurrent and input contributions. The scalar action u that drives the cartpole physics in the first block is then computed from the hidden state as

$$u = \tanh\big(W^{\text{out}} h\big), \qquad W^{\text{out}} \in \mathbb{R}^{1\times 8},$$

which produces the bounded action $u \in [-1, 1]$ used in the force expression $F = F_{\text{mag}} \cdot u$. The exact values of the recurrent weights W^{rec}, input weights W^{in}, biases b, and readout weights W^{out} are defined within the MATLAB function *CartpoleCTRNN.m*.

A.2 Hybrid neural ODE control system benchmarks

A.2.1 ACC

The Adaptive Cruise Control (ACC) benchmark, originally introduced as a neural network controlled system reachability problem in Tran et al. (2020) and adapted to a neural ODE plant model in Manzanas Lopez et al. (2022), is a hybrid neural ODE control system in which an ego vehicle equipped with an ACC controller must maintain a safe distance from a lead vehicle. The system operates in two modes: a *speed control* mode, in which the ego vehicle tracks a driver-specified set speed $v_{\text{set}} = 30$ m/s, and a *spacing control* mode, in which it maintains a safe distance from the lead vehicle. The controller is a feed-forward neural network

with 5 hidden layers of 20 ReLU neurons each and a linear output layer, with a control period of 0.1 s. The verification scenario considers the two vehicles to be initially operating at a safe distance with the ego vehicle in speed control mode, after which the lead vehicle suddenly decelerates at -2 m/s^2. The safety specification requires that the relative distance $D_{\text{rel}} = x_{\text{lead}} - x_{\text{ego}}$ stays above the safe distance $D_{\text{safe}} = D_{\text{default}} + t_{\text{gap}} \cdot v_{\text{ego}}$ throughout a 5 s time horizon, with $D_{\text{default}} = 10$ m and $t_{\text{gap}} = 1.4$ s.

We consider here the version with a neural ODE plant model from Manzanas Lopez et al. (2022), consisting of a 3rd-order 8-dimensional ODE with state vector $x = (x_{\text{lead}}, v_{\text{lead}}, \gamma_{\text{lead}}, x_{\text{ego}}, v_{\text{ego}}, \gamma_{\text{ego}}, a_{\text{ego}}, a_{\text{lead}})^\top \in \mathbb{R}^8$, where x_{lead}, v_{lead}, and γ_{lead} denote the lead vehicle position, velocity, and internal acceleration, x_{ego}, v_{ego}, and γ_{ego} denote the corresponding ego vehicle quantities, and a_{ego}, a_{lead} denote the derivative of the target acceleration commands for the ego and lead vehicles.

The dynamics consists of two kinematic chains, one per vehicle of the following form

$$\dot{x}_{\text{lead}} = v_{\text{lead}}, \quad \dot{v}_{\text{lead}} = \gamma_{\text{lead}}, \quad \dot{x}_{\text{ego}} = v_{\text{ego}}, \quad \dot{v}_{\text{ego}} = \gamma_{\text{ego}},$$

together with a 2-layer feed-forward neural network f_{nn} that drives the four non-kinematic components of $\dot{x}$. The network takes the 4-dimensional sub-state $u = (\gamma_{\text{lead}}, \gamma_{\text{ego}}, a_{\text{ego}}, a_{\text{lead}})^\top$ as input and produces the 4-dimensional output $(\dot{\gamma}_{\text{lead}}, \dot{\gamma}_{\text{ego}}, \dot{a}_{\text{ego}}, \dot{a}_{\text{lead}})^\top = f_{\text{nn}}(u)$. We consider two variants of the plant that differ only in the activation function of f_{nn}.

Linear

In the linear variant, f_{nn} has no activation function and is given by

$$f_{\text{nn}}(u) = W_2(W_1 u + b_1) + b_2,$$

with $W_1 \in \mathbb{R}^{h \times 4}$, $b_1 \in \mathbb{R}^h$, $W_2 \in \mathbb{R}^{4 \times h}$, and $b_2 \in \mathbb{R}^4$, where h denotes the hidden layer width of the parameterizing network (with $h = 20$ in the trained model used in our experiments). Since the hidden layer applies no activation function, f_{nn} reduces to an affine map $f_{\text{nn}}(u) = Au + b$ with $A = W_2 W_1 \in \mathbb{R}^{4 \times 4}$ and $b = W_2 b_1 + b_2 \in \mathbb{R}^4$. The exact values of the weight matrices and bias vectors are loaded from the MATLAB data file *plant_3rd_order_node.mat*.

Nonlinear

In the nonlinear variant, f_{nn} applies a tanh activation between the two layers and is given by

$$f_{\text{nn}}(u) = W_2 \tanh(W_1 u + b_1) + b_2,$$

where W_1, b_1, W_2, b_2 are as in the linear variant and $\tanh(\cdot)$ is applied element-wise to the hidden vector $W_1 u + b_1 \in \mathbb{R}^h$. The exact values of the weight matrices and bias vectors are loaded from the MATLAB data file *plant_3rd_order_tanh.mat*.

A.3 General neural ODE (GNODE) classification benchmarks

A.3.1 MNIST Classification neural ODE

The MNIST classification benchmark uses the standard MNIST handwritten-digit data set (LeCun et al., 1998), embedded in a general neural ODE (GNODE) classifier architecture introduced by Manzanas Lopez et al. (2022), in which an ODE block is added between standard convolutional layers in a feed-forward digit classifier. The input is a 28×28 grayscale image and the output is a 10-dimensional vector of logits, one per digit class. We use the same architecture as in Manzanas Lopez et al. (2022), which combines convolutional pre-processing, a continuous linear ODE block, and a fully-connected output layer.

Architectures. We evaluate the two convolutional variants CNODE_S and CNODE_M, which share a layer-wise architecture as illustrated in Table 4 and differ only in the convolutional channel count. The two convolutions reduce the spatial dimensions from 28×28 at the input to 13×13 at the ODE input, the first convolution does not use padding and produces a 26×26 feature map, and the second convolution uses padding 1 and stride 2 to divide the spatial dimensions to 13×13. After flattening, the ODE input dimension is 676 for CNODE_S and 1690 for CNODE_M.

Table 4: Layer-wise architecture of the MNIST GNODE classifiers CNODE_S and CNODE_M.

Layer	Operation	Output (CNODE_S)	Output (CNODE_M)
1	Conv2D	$26 \times 26 \times 4$	$26 \times 26 \times 10$
2	BatchNorm	$26 \times 26 \times 4$	$26 \times 26 \times 10$
3	ReLU	$26 \times 26 \times 4$	$26 \times 26 \times 10$
4	Conv2D	$13 \times 13 \times 4$	$13 \times 13 \times 10$
5	BatchNorm	$13 \times 13 \times 4$	$13 \times 13 \times 10$
6	ReLU	$13 \times 13 \times 4$	$13 \times 13 \times 10$
7	Flatten	$\mathbb{R}^{676}$	$\mathbb{R}^{1690}$
8	Linear ODE block (see below)	$\mathbb{R}^{676}$	$\mathbb{R}^{1690}$
9	FullyConnected ($\mathbb{R}^n \to \mathbb{R}^{10}$)	$\mathbb{R}^{10}$	$\mathbb{R}^{10}$

ODE block. The ODE block (layer 8) has linear dynamics of the following form

$$\dot{x} \;=\; Ax + b, \qquad x \in \mathbb{R}^n,$$

with $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$. The values of A, b, the convolutional weights, the batch-normalization parameters, and the output-layer weights are obtained by loading the trained MATLAB data files *odecnn_mnist_tiny.mat* for CNODE_S and *odecnn_mnist_mid2.mat* for CNODE_M.

A.3.2 Hybrid reachability pipeline

Verification of these GNODE classifiers requires computing the reachability throughout the entire network. Since the pre-ODE convolutional and ReLU layers and the post-ODE fully-connected layer are discrete neural network layers that TNODEV does not support natively, we therefore construct a hybrid pipeline that uses NNV 2.0 ImageStar reachability for the discrete pre-ODE layers (layers 1–7), TNODEV's reachability block for the linear ODE block (layer 8), and NNV 2.0 Star set reachability for the final fully-connected layer (layer 9).

TNODEV main reachability block is based on CTMM as discussed in Section 3.2, and CTMM is the backend used for all benchmarks in Section 5.2. For linear ODE block of the MNIST GNODE classifier specifically, we instead use a closed-form matrix-exponential propagator that gives the exact linear flow over the integration interval $[0, t_f]$, followed by the standard interval analysis construction (Moore et al., 2009) that produces the tightest axis-aligned box enclosing the affine image of a box.

We attempted to apply CTMM to the linear ODE block of this benchmark on a full image L_∞ attack (i.e., all 784 pixels are perturbed), but CTMM produced no verified images on either network within the refinement iteration K and wall-clock timeout $T_{\max}$, and only on a single pixel attack on individual images did we obtain verified verdicts in approximately 26 minutes per image. We attribute this to the wrapping effect of integrating the embedded $2n$-dimensional monotone system over the time horizon on a large, densely coupled A. The closed-form matrix-exponential propagator avoids this by encoding the exact linear flow of $\dot{x} = Ax + b$ over $[0, t_f]$ in a single step, with no time integration of an embedded system. A similar matrix-exponential construction is implemented in NNV 2.0 `LinearODE` class and is used by Manzanas Lopez et al. (2022) for the linear ODE blocks GNODE benchmarks. The difference between the two pipelines is in what is propagated through this construction, as NNV 2.0 propagates a predicate-constrained Star set, whereas TNODEV propagates an axis-aligned box.

We now make the construction precise and establish its soundness. We state two lemmas characterizing the two pieces of the construction (the closed-form propagator and the tightest box enclosure of the affine image of a box), and combine them in a sound theorem.

Lemma 1 (Augmented matrix exponential propagator)**.** *Let $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $t_f \in \mathbb{R}$. Define the augmented matrix*

$$\bar{A} \;=\; \begin{bmatrix} A & b \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)},$$

and let $E = \exp(\bar{A}\, t_f)$. Then E has the block form

$$E \;=\; \begin{bmatrix} M & c \\ 0 & 1 \end{bmatrix}$$

for some $M \in \mathbb{R}^{n\times n}$ and $c \in \mathbb{R}^n$. In addition, for every initial condition $x_0 \in \mathbb{R}^n$, the unique solution $x(\cdot)$ of the linear ODE $\dot{x} = Ax + b$ with $x(0) = x_0$ satisfies

$$x(t_f) \;=\; Mx_0 + c.$$

Proof. We first establish the block structure of E. The bottom row of $\bar{A}$ is zero, so for every series index $j \geq 1$ the bottom row of $\bar{A}^j$ is zero (the bottom row of a matrix product is a weighted some of the bottom row of its left factor). The matrix exponential admits the absolutely convergent series expansion $\exp(\bar{A}\, t_f) = \sum_{j=0}^{\infty} \frac{(\bar{A}\, t_f)^j}{j!}$, in which only the $j = 0$ term contributes a nonzero bottom row, namely $[0, \ldots, 0, 1]$ from the identity. Hence E has bottom row $[0, \ldots, 0, 1]$, giving the claimed block form with $M = E_{[1:n,1:n]}$ and $c = E_{[1:n,n+1]}$.

We now derive the solution formula. Define the augmented state $z(t) := (x(t)^\top, 1)^\top \in \mathbb{R}^{n+1}$. Differentiating gives

$$\dot{z}(t) \;=\; \begin{pmatrix} \dot{x}(t) \\ 0 \end{pmatrix} \;=\; \begin{pmatrix} Ax(t) + b \\ 0 \end{pmatrix} \;=\; \begin{bmatrix} A & b \\ \mathbf{0} & 0 \end{bmatrix} \begin{pmatrix} x(t) \\ 1 \end{pmatrix} \;=\; \bar{A} z(t),$$

so z satisfies a homogeneous linear ODE with constant matrix $\bar{A}$. The unique solution of this ODE with initial condition $z(0) = (x_0^\top, 1)^\top$ is $z(t_f) = \exp(\bar{A}\, t_f)\, z(0) = E\, (x_0^\top, 1)^\top$. Reading off the top n components yields $x(t_f) = Mx_0 + c$. □

The construction in Lemma 1 relies on the standard affine-to-homogeneous lift of a linear time-invariant system (Antsaklis & Michel, 2007), in which a constant unit coordinate is appended to the state so that the affine system $\dot{x} = Ax + b$ becomes the homogeneous system $\dot{z} = \bar{A}z$ for the augmented state $z = (x^\top, 1)^\top$. The matrix M defined in Lemma 1 then coincides with the state transition matrix $\exp(A\, t_f)$, and the offset c coincides with the variation of constants integral $\int_0^{t_f} \exp(A(t_f - s))\, b\, ds$. The augmented matrix construction recovers both quantities from a single matrix exponential of $\bar{A}$, and no explicit numerical integration is required.

Lemma 2 (Tightest box enclosure of the affine image of a box)**.** *Let $M \in \mathbb{R}^{n\times n}$, $c \in \mathbb{R}^n$, and let $[\underline{x}, \overline{x}] \subseteq \mathbb{R}^n$ with $\underline{x} \leq \overline{x}$ component-wise. Define $M_+ = \max(M, \mathbf{0})$ and $M_- = \min(M, \mathbf{0})$ element-wise. Then the smallest axis-aligned box $[\underline{y}, \overline{y}]$ containing the set $\{Mx + c : x \in [\underline{x}, \overline{x}]\}$ is given by*

$$\underline{y} \;=\; M_+\underline{x} + M_-\overline{x} + c, \qquad \overline{y} \;=\; M_+\overline{x} + M_-\underline{x} + c.$$

Proof. For a fixed row index $i \in \{1, \ldots, n\}$, the i-th component of the affine map is $(Mx+c)_i = \sum_{j=1}^{n} M_{ij}\, x_j + c_i$. The terms decouple across j, so the minimum of $(Mx+c)_i$ over $x \in [\underline{x}, \overline{x}]$ is the sum of the per-coordinate minima of $M_{ij}\, x_j$ over $x_j \in [\underline{x}_j, \overline{x}_j]$, plus c_i. We compute these per-coordinate minima as follows:

- If $M_{ij} \geq 0$, then $M_{ij}\, x_j$ is non-decreasing in x_j, so its minimum over $[\underline{x}_j, \overline{x}_j]$ is $M_{ij}\, \underline{x}_j$. In this case $(M_+)_{ij} = M_{ij}$ and $(M_-)_{ij} = 0$, so $M_{ij}\, \underline{x}_j = (M_+)_{ij}\, \underline{x}_j + (M_-)_{ij}\, \overline{x}_j$.
- If $M_{ij} < 0$, then $M_{ij}\, x_j$ is decreasing in x_j, so its minimum over $[\underline{x}_j, \overline{x}_j]$ is $M_{ij}\, \overline{x}_j$. In this case $(M_+)_{ij} = 0$ and $(M_-)_{ij} = M_{ij}$, so $M_{ij}\, \overline{x}_j = (M_+)_{ij}\, \underline{x}_j + (M_-)_{ij}\, \overline{x}_j$.

In both cases the per-coordinate minimum equals $(M_+)_{ij}\, \underline{x}_j + (M_-)_{ij}\, \overline{x}_j$. Summing over j and adding c_i yields

$$\min_{x\in[\underline{x},\overline{x}]} (Mx + c)_i \;=\; \sum_{j=1}^{n} \left[(M_+)_{ij}\, \underline{x}_j + (M_-)_{ij}\, \overline{x}_j\right] + c_i \;=\; (M_+\underline{x} + M_-\overline{x} + c)_i \;=\; \underline{y}_i.$$

A symmetric argument with minima replaced by maxima yields $\max_{x\in[\underline{x},\overline{x}]}(Mx+c)_i = (M_+\overline{x}+M_-\underline{x}+c)_i = \overline{y}_i$. Both extrema are attained by points in $[\underline{x},\overline{x}]$ (specifically, by the corners identified above, which may differ for different rows i). Therefore $[\underline{y},\overline{y}]$ contains the image set and no smaller axis-aligned box has this property. □

We can now combine the two lemmas to establish the soundness of the linear ODE interval reachability used by TNODEV on this benchmark.

Theorem 1 (Soundness of the closed-form interval reachability for the linear ODE block)**.** *Let $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $t_f \in \mathbb{R}$, and let $[\underline{x}_0, \overline{x}_0] \subseteq \mathbb{R}^n$. Let M, c be as in Lemma 1 and M_+, M_- as in Lemma 2. Define*

$$\underline{x}_{t_f} \;=\; M_+\underline{x}_0 + M_-\overline{x}_0 + c, \qquad \overline{x}_{t_f} \;=\; M_+\overline{x}_0 + M_-\underline{x}_0 + c.$$

Then for every initial condition $x_0 \in [\underline{x}_0, \overline{x}_0]$, the unique solution $x(\cdot)$ of $\dot{x} = Ax + b$ with $x(0) = x_0$ satisfies $x(t_f) \in [\underline{x}_{t_f}, \overline{x}_{t_f}]$. Moreover, $[\underline{x}_{t_f}, \overline{x}_{t_f}]$ is the smallest axis-aligned box containing the reachable set $\{\Phi(t_f, x_0) : x_0 \in [\underline{x}_0, \overline{x}_0]\}$.

Proof. Let $x_0 \in [\underline{x}_0, \overline{x}_0]$ be an arbitrary initial condition, and let $x(\cdot)$ denote the unique solution of $\dot{x} = Ax+b$ with $x(0) = x_0$. By Lemma 1, $x(t_f) = Mx_0 + c$, so $x(t_f) \in \mathcal{R} := \{Mx + c : x \in [\underline{x}_0, \overline{x}_0]\}$. By Lemma 2, the smallest axis-aligned box containing $\mathcal{R}$ is exactly $[\underline{x}_{t_f}, \overline{x}_{t_f}]$. Hence $x(t_f) \in [\underline{x}_{t_f}, \overline{x}_{t_f}]$, and no smaller axis-aligned box can contain the reachable set $\mathcal{R}$ since by Lemma 1 the reachable set is in bijection with the affine image $\{Mx_0 + c : x_0 \in [\underline{x}_0, \overline{x}_0]\}$. □

The output bounds in Theorem 1 are evaluated in a single pass, as computing $\underline{x}_{t_f}$ and $\overline{x}_{t_f}$ requires one matrix exponential (precomputed once per network) and two matrix-vector products per image (two for the lower bound and two for the upper bound). The output interval $[\underline{x}_{t_f}, \overline{x}_{t_f}]$ is the tightest sound axis-aligned over-approximation of the reachable set. The reachable set itself is in general a zonotope-like polytope (the affine image of an axis-aligned box under M) and not exactly a box. This is precisely why TNODEV's interval representation is lossy on this benchmark compared to NNV 2.0 Star set representation, which can capture the polytope structure exactly.

A.3.3 Star-set to interval conversion at the ODE input

The pre-ODE layers (layers 1–7) produce a Star set representation R_{pre} with predicate variables α constrained by linear inequalities $C\alpha \le d$. NNV 2.0 approximate ReLU reachability adds these inequality constraints whenever a neuron's input range contains both positive and negative values, the constraints carry information that the interval box hull alone does not. Since our closed-form propagator operates on intervals, the pipeline involves a Star set → interval → Star set conversion before and after the ODE block boundary. We extract per-coordinate bounds $[\underline{x}_{\text{pre}}, \overline{x}_{\text{pre}}]$ from R_{pre} at the ODE input, propagate them through the closed-form propagator (Theorem 1) to obtain $[\underline{x}_{\text{ode}}, \overline{x}_{\text{ode}}]$, and then re-encode this interval as a Star set (specifically, a Star set whose predicate variables are bounded by the box $[\underline{x}_{\text{ode}}, \overline{x}_{\text{ode}}]$) so that NNV 2.0 exact Star set reachability can run on the final fully-connected layer.

The forward conversion (Star set → interval) discards the predicate constraints $C\alpha \le d$. We provide two sound modes that differ in whether they exploit these constraints when extracting bounds:

- *LP-tight* uses NNV 2.0 `getRanges()`, which solves one linear program per output coordinate to obtain the tightest sound per-coordinate bounds that respects the predicate constraints $C\alpha \le d$. The mode inherits NNV 2.0 reliance on `linprog`, which we observed to occasionally encounter degenerate LP geometries on individual Star sets.

- *Fast* uses NNV 2.0 `getBoxFast()`, which ignores the predicate constraints and treats α as a hyper-rectangle. The resulting bounds remain sound but are looser whenever predicate constraints are active.

The reverse conversion (interval → Star set) is direct: given the post-propagator interval $[\underline{x}_{\text{ode}}, \overline{x}_{\text{ode}}]$, we construct a Star set with no inequality predicate constraints whose predicate variables vary independently within these bounds.

We report both forward-conversion modes in Table 3 to separate the contribution of bound-extraction tightness at the boundary from the interval-propagation lossiness incurred by the closed-form propagator. On this benchmark the two modes return identical verdict counts, indicating that the interval over-approximation through the linear ODE is the binding constraint rather than the predicate-constraint information at the boundary.

Iterative refinement. Neither of the two tools uses iterative refinement for this benchmark. Indeed, NNV 2.0 does not currently implement input-set refinement, so to keep the comparison meaningful we also use TNODEV without refinements. We note that if refinement is added, it will be applied to the global GNODE input set rather than to the ODE block input only, even though NNV 2.0 Star set propagation through $\exp(At_f)$ is exact at the ODE layer, the pre-ODE convolutional and ReLU layers use approximate reachability, so exactness at the ODE layer alone does not make global refinement unnecessary. For TNODEV, refinement on the global GNODE input would split the input set into sub-sets, propagate each through the full hybrid pipeline, and take the union of the resulting verdicts. We leave this global input-set refinement strategy as a future work.

A.3.4 Where the verification gap comes from

The 20-image gap between TNODEV and NNV 2.0 originates entirely in the linear ODE block. NNV 2.0 propagates the predicate-constrained Star set representation through the linear ODE exactly, while TNODEV interval propagation produces the bounding box of the affine image of the input box (Theorem 1): a sound enclosure, but a strict super-set of the actual image set whenever the input box's image under M is not itself a box. The characteristic worst-case widening of this enclosure can be summarized by the propagator's L_∞ norm: $\|M\|_\infty \approx 10.5$ on CNODE_S and $\|M\|_\infty \approx 14.1$ on CNODE_M, where M is the propagator matrix from Lemma 1. The quantity $\|M\|_\infty$ is the maximum row sum of $|M|$; if every input coordinate has width δ then the maximum output coordinate width is bounded by $\|M\|_\infty \cdot \delta$. This is a diagnostic characterization of the interval hull wrapping, not an additional factor applied during reachability, and the interval propagation itself is a single closed-form evaluation. The actual aggregate width expansion observed on representative test images is roughly 7× on CNODE_S ($\sum w_{\text{post}} / \sum w_{\text{pre}} = 6.91$) and 8× on CNODE_M (ratio 8.36) at $\epsilon = 0.5/255$. After this widening, NNV2.0 tighter post-ODE Star set representation still produces logit bounds where the target-class lower bound dominates the other classes' upper bounds on nearly every test image (the condition required for ϵ-robustness). TNODEV looser post-ODE bounding box only achieves this dominance on the more confidently classified images.

A.3.5 Applicability of TNODEV on this benchmark

TNODEV is sound and applicable across this benchmark in the sense that the hybrid pipeline runs end-to-end on every test image and produces a sound (over-approximation) verdict. However, the interval-based propagation through the linear ODE block makes TNODEV best suited to small adversarial perturbations on this class of GNODE classifiers, where the interval hull enclosure of the linear ODE image remains tight enough to satisfy the per-class dominance condition required for ϵ-robustness. At $\epsilon = 0.5/255$ (Table 3) TNODEV verifies 58–60% of test images. At $\epsilon = 1/255$, twice this magnitude, the verification rate falls to 1/50 on both networks under both bound-extraction modes, while NNV 2.0 retains 49/50 on CNODE_S and 50/50 on CNODE_M (Manzanas Lopez et al., 2022). For verification under larger adversarial perturbations on this class of linear ODE based classifiers, NNV 2.0 Star set representation remains a more appropriate tool choice than TNODEV interval based propagation.

B Example Results

B.1 Reachability comparison of TNODEV against NNV 2.0 and CORA

This appendix provides more details on the comparison of the reachability analysis results from Section 5.1 between TNODEV, NNV 2.0, and CORA on the Spiral nonlinear and FPA benchmarks. For each tool, we measure the wall-clock time to compute a single reachable set over-approximation Ω from the initial input set $\mathcal{X}_{in}$, together with the size of the resulting over-approximation. The size metric we report is the *geometric mean width*:

$$\mu(\Omega) \;=\; \sqrt[n]{\mathrm{Vol}(\Omega)}, \tag{3}$$

where n is the state dimension and $\mathrm{Vol}(\Omega)$ is the volume of the over-approximation. The n-th root makes the metric dimensionally comparable across benchmarks of different state dimensions, i.e., $\mu(\Omega)$ has the same units as a single state coordinate, and is exactly the side length of the n-dimensional cube of the same volume as Ω. Smaller $\mu(\Omega)$ indicates a tighter over-approximation. For an axis-aligned interval over-approximation $\Omega = [\underline{x}, \overline{x}]$, equation 3 simplifies to the geometric mean of the component widths:

$$\mu(\Omega) \;=\; \Big(\prod_{i=1}^{n}(\overline{x}_i - \underline{x}_i)\Big)^{1/n}. \tag{4}$$

Implementation per tool. The three tools represent the reachable set differently, so the volume in equation 3 is computed differently for each of them:

- **TNODEV** (intervals): Ω is an axis-aligned interval and $\mathrm{Vol}(\Omega) = \prod_{i=1}^{n}(\overline{x}_i - \underline{x}_i)$.
- **CORA** (zonotopes): Ω is a zonotope and $\mathrm{Vol}(\Omega)$ is obtained from CORA built-in `volume()` function, which sums the parallelotope contributions induced by the zonotope's generators.
- **NNV 2.0** (Star set): Ω is a Star set and $\mathrm{Vol}(\Omega)$ is computed as the volume of the tightest axis-aligned box enclosing the Star set, obtained from NNV 2.0 built-in `getBox()` function.[2]

Spiral nonlinear results. Table 5 reports the comparison on the Spiral nonlinear 2D benchmark. TNODEV is **14** times faster than NNV 2.0 and **83** times faster than CORA, while the over-approximations produced by NNV 2.0 and CORA are approximately **4** and **4.3** times tighter than TNODEV. This reflects the cost of axis-aligned interval propagation when the actual reachable set is not axis-aligned, i.e., the spiral dynamics rotate the input box and the resulting interval over-approximation widens substantially because it must contain the entire rotated set in an axis-aligned bounding box, an effect that NNV 2.0 and CORA mitigate through their more complex set representations.

Table 5: Reachability comparison on Spiral nonlinear.

Tool	**Set representation**	$\mu(\Omega)$	**Time (s)**
TNODEV	Intervals	2.031	0.30
NNV 2.0	Star set	0.507	4.20
CORA	Zonotopes	0.467	24.99

FPA results. Table 6 reports the comparison on the FPA 5D benchmark. TNODEV is approximately **7** times faster than NNV 2.0 and **39** times faster than CORA, while TNODEV interval over-approximation is approximately **1.3** times tighter than NNV 2.0 Star set, while CORA's zonotope remains the tightest at approximately **1.4** times tighter than TNODEV.

[2]Exact polytope-volume computation in higher dimensions is prohibitively expensive (Dyer & Frieze, 1988; Khachiyan, 1989), so the box-hull volume is used uniformly across both benchmarks for the NNV 2.0.

Table 6: Reachability comparison on FPA.

Tool	Set representation	$\mu(\Omega)$	Time (s)
TNODEV	Intervals	0.030	0.32
NNV 2.0	Star set	0.040	2.16
CORA	Zonotopes	0.022	12.34

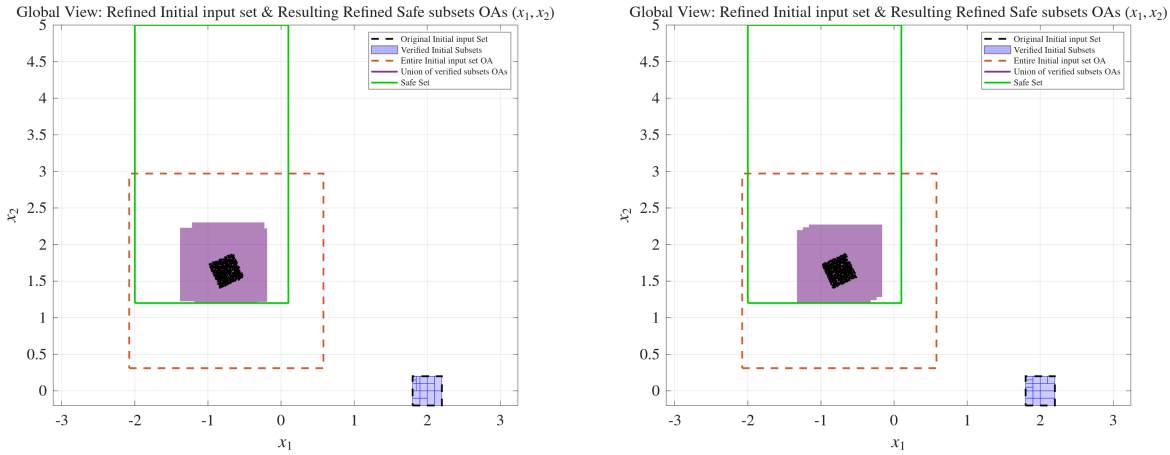
Across the two benchmarks, the reachability comparison exposes a consistent trade-off: TNODEV provides the lowest per-call wall-clock time, at the cost of wider over-approximations when the reachable set is far from axis-aligned. The Spiral nonlinear benchmark illustrates a setting where intervals are not the natural set representation, as rotational dynamics produce a reachable set that is poorly approximated by any axis-aligned box. The FPA benchmark illustrates a setting where the reachable set is closer to an axis-aligned box, and interval propagation is competitive with the box hull of richer representations. The verifier architecture of Section 4 offers a solution for the Spiral nonlinear case, as instead of relying on a single reachability call to be tight, TNODEV wraps a fast reachability backend based on CTMM in the verification and refinement loop, splitting the input set and re-computing the reachability on each sub-cell until each sub-cell's over-approximation becomes tight enough for the property to be certified, or the refinement iteration limit is reached.

B.2 Spiral Linear

The safety specification is:

- *Initial set:* $\mathcal{X}_{in} = [1.8, 2.2] \times [-0.2, 0.2]$
- *Safe set:* $\mathcal{X}_s = [-2,\ 0.1] \times [1.2,\ 5]$

The initial set follows the reachability setup of Manzanas Lopez et al. (2022), while the safe set is defined by us for this verification task. Figure 3 shows the verified subsets obtained with the naive and MSIR heuristics, both certify the specification as `SAFE`, with the union of the violet verified subsets covering the initial set and remaining inside the green safe set.

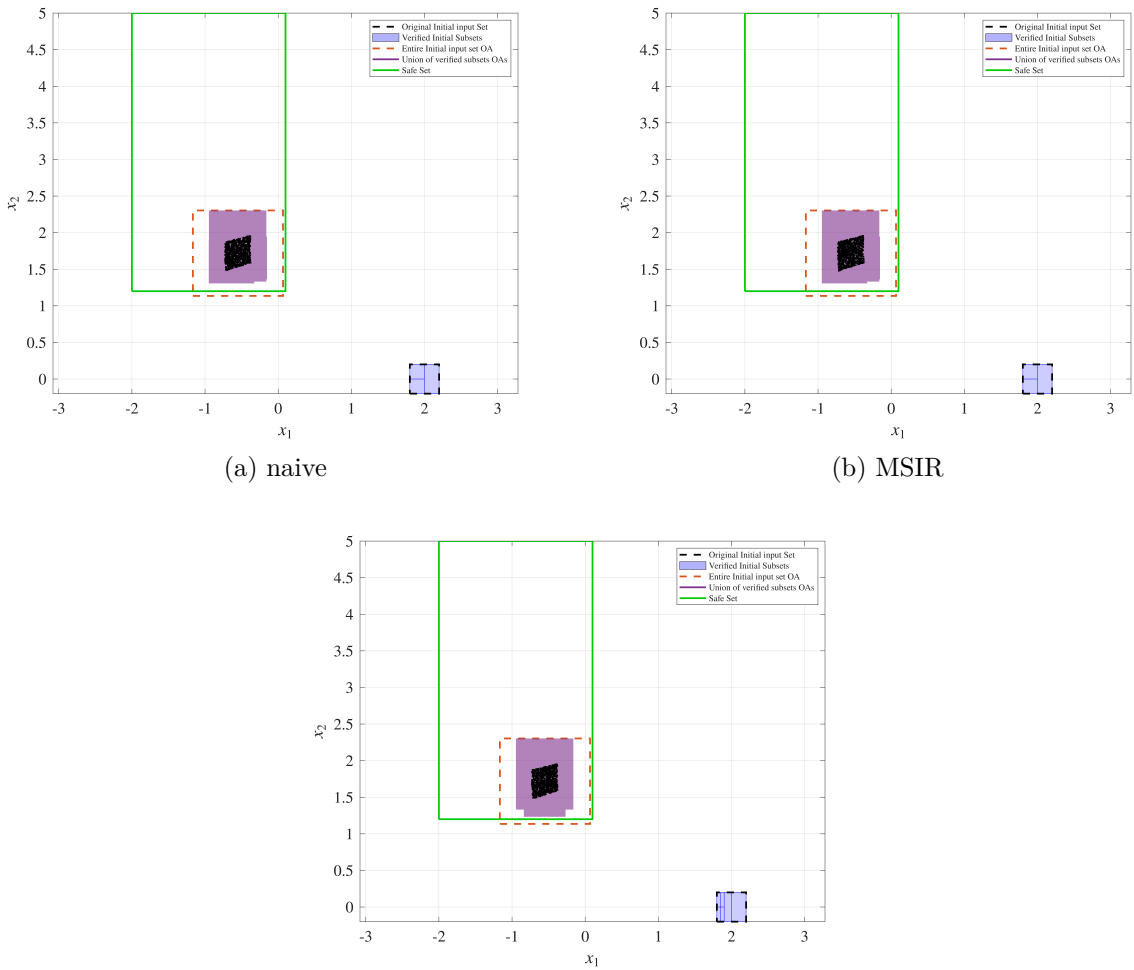


(a) naive (b) MSIR

Figure 3: TNODEV verification results on the linear spiral 2D benchmark using naive and MSIR refinement. In all sub figures, the dashed red rectangle is the over-approximation $\Omega(\mathcal{X}_{in})$ from a single reachability call on the unsplit initial set, the green rectangle is the safe set $\mathcal{X}_s$, the violet filled rectangles represent the union of the verified subsets, and the black dots are sampled successor trajectories.

B.3 Spiral nonlinear

The safety specification is similar to spiral linear in Appendix B.2. Here as well, the initial set follows the reachability setup of Manzanas Lopez et al. (2022), while the safe set is defined by us for this verification task. Figure 4 shows the verified subsets obtained with the three refinement heuristics. The naive and MSIR heuristics certify the specification `SAFE`; ING also certifies it but requires more refinement iterations.



(a) naive

(b) MSIR

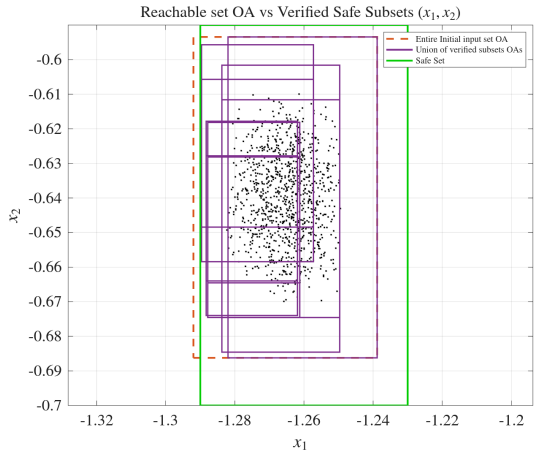
(c) ING

Figure 4: TNODEV verification results on the nonlinear spiral 2D benchmark using naive, MSIR and ING refinement. In all sub figures, the dashed red rectangle is the over-approximation $\Omega(\mathcal{X}_{in})$ from a single reachability call on the unsplit initial set, the green rectangle is the safe set $\mathcal{X}_s$, the violet filled rectangles represent the union of the verified subsets, and the black dots are sampled successor trajectories.

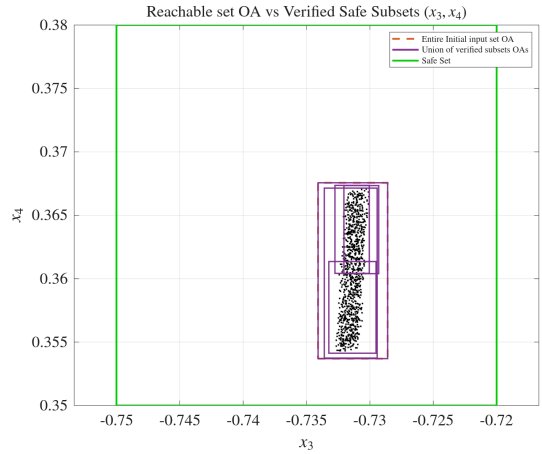
B.4 FPA

The safety specification is:

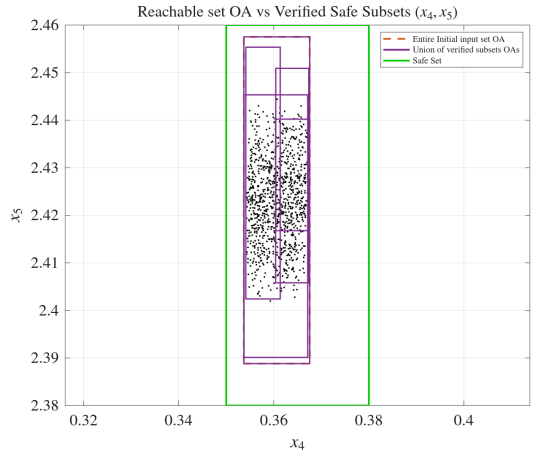
- *Initial set:* $\mathcal{X}_{in} = [-0.01, 0.01] \times [-0.59587, -0.57587] \times [0.79, 0.81] \times [0.51323, 0.53323] \times [0.69, 0.71]$
- *Safe set:* $\mathcal{X}_s = [-1.29, -1.23] \times [-0.70, -0.59] \times [-0.75, -0.72] \times [0.35, 0.38] \times [2.38, 2.46]$



(a) $x_1 - x_2$

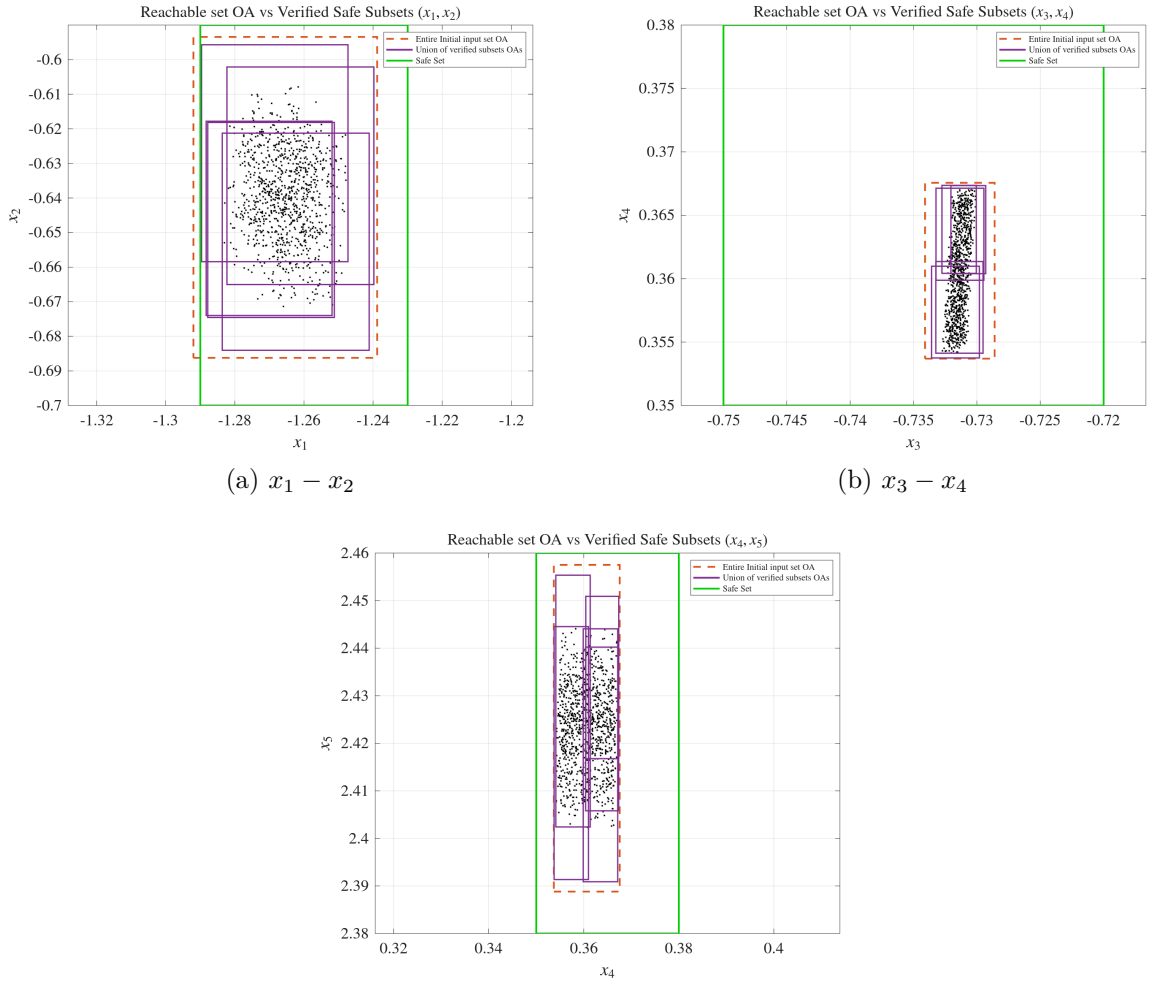


(b) $x_3 - x_4$



(c) $x_4 - x_5$

Figure 5: TNODEV verification results on the FPA 5D benchmark using naive refinement. In all sub figures, the dashed red rectangle is the over-approximation $\Omega(\mathcal{X}_{in})$ from a single reachability call on the unsplit initial set, the green rectangle is the safe set $\mathcal{X}_s$, the violet rectangles represent the verified subsets, and the black dots are sampled successor trajectories.

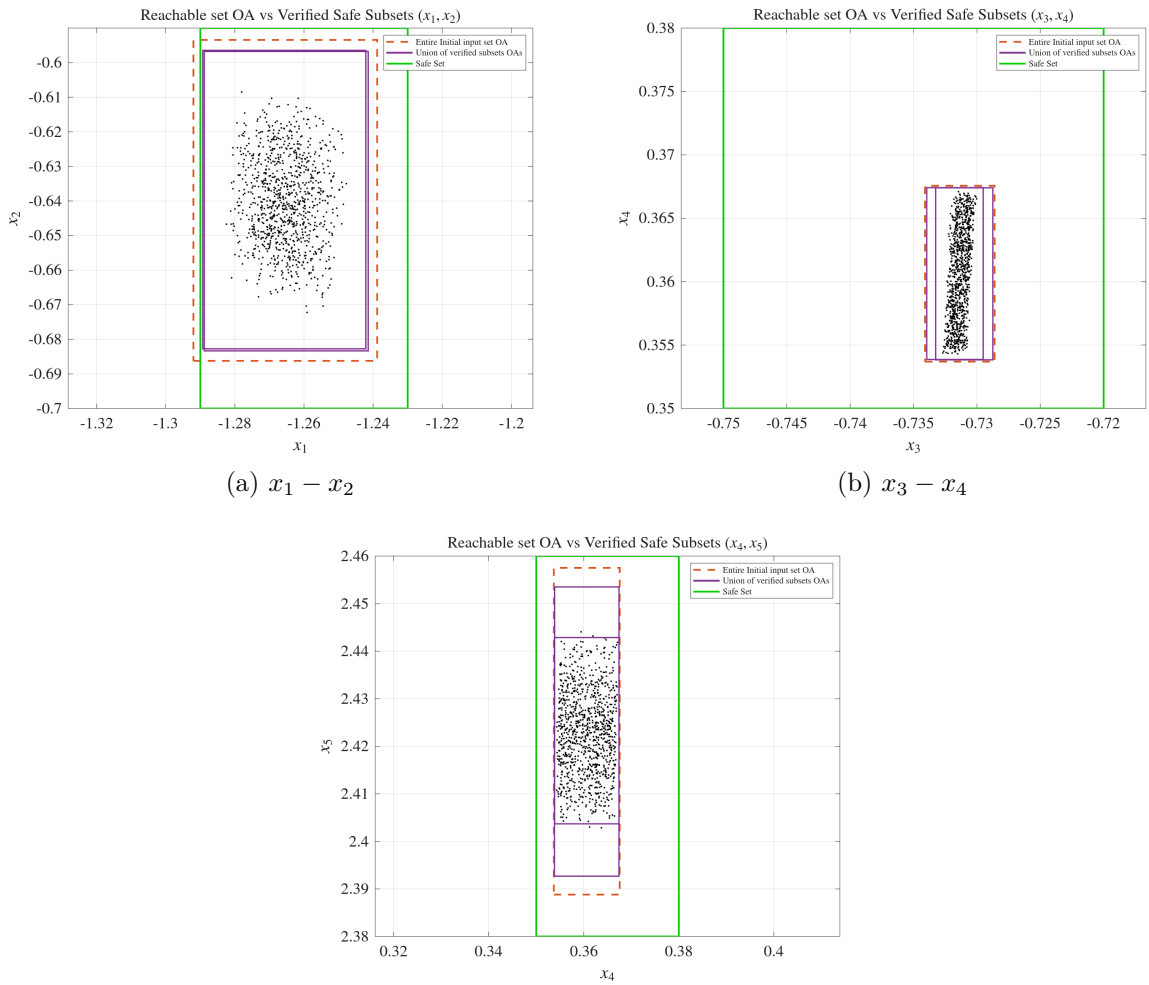


(a) $x_1 - x_2$

(b) $x_3 - x_4$

(c) $x_4 - x_5$

Figure 6: TNODEV verification results on the FPA 5D benchmark using MSIR refinement. In all sub figures, the dashed red rectangle is the over-approximation $\Omega(\mathcal{X}_{in})$ from a single reachability call on the unsplit initial set, the green rectangle is the safe set $\mathcal{X}_s$, the violet rectangles represent the verified subsets, and the black dots are sampled successor trajectories.



(a) $x_1 - x_2$

(b) $x_3 - x_4$

(c) $x_4 - x_5$

Figure 7: TNODEV verification results on the FPA 5D benchmark using ING refinement. In all sub figures, the dashed red rectangle is the over-approximation $\Omega(\mathcal{X}_{in})$ from a single reachability call on the unsplit initial set, the green rectangle is the safe set $\mathcal{X}_s$, the violet rectangles represent the verified subsets, and the black dots are sampled successor trajectories.

Figures 5–7 show the verified subsets for the naive, MSIR, and ING heuristics, projected onto the three planes x_1–x_2, x_3–x_4, and x_4–x_5. All three heuristics certify the specification `SAFE`.

B.5 Cartpole

The safety specification is:

- *Initial set:* $\mathcal{X}_{in} = [x_0 - 10^{-4}, x_0 + 10^{-4}]$ with center $x_0 = (0, 0, 0.001, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, i.e., a small initial pole angle $\theta_0 = 0.001$ with all other states at zero.

- *Safe set:* $\mathcal{X}_s = [-0.015, 0.015] \times [-0.110, 0.060] \times [-0.015, 0.020] \times [-0.050, 0.160] \times [-0.2, 0.2]^8$, bounding the cart position, cart velocity, pole angle, pole angular velocity, and the eight hidden states.

B.6 ACC

The safety specification is:

- *Initial set:* $x_{\text{lead}} \in [90, 110]$, $v_{\text{lead}} \in [31, 33]$, $x_{\text{ego}} \in [10, 11]$, $v_{\text{ego}} \in [29, 31]$, with the remaining plant states initialized as in Manzanas Lopez et al. (2022).
- *Safe set:* $D_{\text{rel}} \geq D_{\text{safe}}$ with $D_{\text{default}} = 10$ m, $t_{\text{gap}} = 1.4$ s, and $\alpha = 1$, evaluated over the 5 s horizon.

The safe set specification is taken directly from the original ACC benchmark of Tran et al. (2020). The velocity ranges of the initial set are widened relative to the nominal benchmark initialization in order to exercise the verification and refinement loop, as under the nominal narrow initial set, the specification is verified by a single reachability call without triggering any refinements.

C Cost analysis of refinement heuristics

This appendix details the computational cost of the three refinement heuristics introduced in Section 4.3. We measure cost in *reachability calls per refinement decision*, which is the dominant operation in the verifier, since each reachability call integrates a $2n$-dimensional embedded ODE over the specified time horizon and is orders of magnitude more expensive than any other per-cell operation.

Baseline. A refinement decision is reached only after a failed verification check on the current cell $\mathcal{X}$. By the time refinement runs, the verifier has already spent one reachability call to obtain $\Omega(\mathcal{X})$. The cost statements below count only *additional* calls beyond this baseline.

Naive. Picks $i^\star = \arg\max_i r_i$, using only the cell's radius vector r. The radius is a property of the cell, known without any computation. Cost: no additional reachability calls and no other meaningful computation.

MSIR. Computes $s_i = r_i \cdot \max_k \max(|\underline{J}_{ki}|, |\overline{J}_{ki}|)$, which requires the radius (already known) and interval Jacobian bounds $[\underline{J}, \overline{J}]$ over the cell. The Jacobian bounds are obtained by a single symbolic pass through the network's analytic Jacobian using interval arithmetic, which is a different operation from a reachability call. To make the contrast more clear: a reachability call evaluates the network's forward pass thousands of times during the ODE integration, whereas computing the interval Jacobian requires only a single interval-bound pass through the network's layers. Cost: no additional reachability calls, and one Jacobian-bounding call per decision.

ING. Estimates $g_j \approx \partial w / \partial x_j$ by finite differences. Computing g_j requires $w(\mathcal{X}_{j,\delta})$, the over-approximation width of the cell perturbed by δ along dimension j. Since the reachability call evaluates complex, non-linear ODE dynamics numerically over a time interval, the resulting over-approximation lacks a closed-form analytic expression. Therefore, its exact derivative cannot be calculated directly, and $w(\mathcal{X}_{j,\delta})$ can only be obtained by running the full reachability call on the perturbed cell. One such call is needed per input dimension. Cost: n additional reachability calls per cell being refined.

Summary. Table 7 summarizes the additional computational overhead required by each heuristic whenever a cell needs to be split:

Heuristic	Extra reach calls	Other cost
Naive	0	—
MSIR	0	1 Jacobian-bounding call
ING	n	—

Table 7: Cost of each refinement heuristic per refinement decision, in addition to the one reachability call already spent on the failed verification check.

Practical implications. Reachability call cost dominates the total verification time, so ING n-fold overhead per decision accumulates rapidly across all refinement iterations. Furthermore, our empirical results in Table 2 show that ING performance is highly inconsistent. Because ING relies on local finite-difference estimates, and it can sometimes find highly efficient splits (as seen in the FPA benchmark). However, this localized view can also lead to poor splitting choices overall, causing the verifier to time out. The effect is highly benchmark-dependent:

- **Spiral linear ($n = 2$).** Even at low dimensions, ING completely fails to converge for the case of Spiral linear, after 5000 iterations (13.16 min) with an `UNKNOWN` verdict. In contrast, Naive and MSIR easily verify the system in around 29 iterations in 6.5 seconds. This highlights that ING's local gradient estimates can get stuck in sub-optimal splitting strategies for certain dynamics.
- **Spiral nonlinear ($n = 2$).** ING converges but requires significantly more iterations (11) and time (46.95 s) than both MSIR (3 iterations, 10.78 s) and Naive (5 iterations, 15.60 s).
- **FPA ($n = 5$).** This benchmark demonstrates ING's theoretical appeal, as ING achieves the lowest number of required iterations (only 3, compared to 17 for Naive and 9 for MSIR). However, because each of those decisions costs 5 additional reachability calls, ING's total wall-clock time (53.78 s) remains higher than MSIR (34.50 s). MSIR ultimately provides the most optimal balance between minimizing iterations and maintaining a low per-decision cost.
- **Cartpole ($n = 12$).** MSIR shows its clear scalability advantage here, drastically reducing iterations compared to Naive (63 vs. 2037) and cutting the total verification time from 2.73 min down to 56.43 s. Meanwhile, ING's compounding 12× reachacility calls overhead, combined with potentially poor local splits, causes it to reach the refinement iteration limit.
- **ACC ($n = 8$).** Both plant variants are verified `SAFE` by naive and MSIR, but ING reaches the wall-clock timeout on both without verifying any subset, returning `UNKNOWN`. This reinforces the pattern that ING's n-fold per-decision overhead makes it the least reliable heuristic on the higher-dimensional benchmarks. On the linear variant MSIR is faster than naive (2.72 vs 4.78 min), while on the nonlinear variant MSIR processes fewer iterations than naive (37 vs 41) but does not reduce wall-clock time (29.61 vs 20.03 min), one of the few cases where MSIR iterations reduction do not translate into a time saving.

The empirical takeaway is that **MSIR is situated as the optimal default heuristic** for TNODEV. Its Jacobian-bounding overhead is practically negligible relative to a reachability call, yet it reliably reduces iteration counts and verification times compared to naive heuristic. ING, while conceptually interesting and capable of highly precise splits in specific scenarios (like FPA in Figure 7), suffers from prohibitive n-fold scaling costs and unpredictable convergence stability (as seen in Spiral linear, Cartpole, and ACC variants), making it impractical for general use and only case by case dependent.