CERTIFIED TRAINING WITH BRANCH-AND-BOUND: A CASE STUDY ON LYAPUNOV-STABLE NEURAL CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

We study the problem of learning Lyapunov-stable neural controllers which provably satisfy the Lyapunov asymptotic stability condition within a region-of-attraction. Compared to previous works which commonly used counterexample guided training on this task, we develop a new and generally formulated certified training framework named **CT-BaB**, and we optimize for differentiable verified bounds, to produce verification-friendly models. In order to handle the relatively large region-of-interest, we propose a novel framework of training-time branch-and-bound to dynamically maintain a training dataset of subregions throughout training, such that the hardest subregions are iteratively split into smaller ones whose verified bounds can be computed more tightly to ease the training. We demonstrate that our new training framework can produce models which can be more efficiently verified at test time. On the largest 2D quadrotor dynamical system, verification for our model is more than 5X faster compared to the baseline, while our size of region-of-attraction is 16X larger than the baseline.

1 INTRODUCTION

Deep learning techniques with neural networks (NNs) have greatly advanced abundant domains in recent years. Despite the impressive capability of NNs, it remains challenging to obtain provable guarantees on the behaviors of NNs, which is critical for the trustworthy deployment of NNs especially in safety-critical domains. One area of particular concern is safe control for robotic systems with NN-based controllers (Chang et al., 2019; Dai et al., 2021; Wu et al., 2023; Yang et al., 2024). There are many desirable properties in safe control, such as reachability w.r.t. target and avoid sets (Althoff & Kochdumper, 2016; Bansal et al., 2017; Dutta et al., 2019; Everett et al., 2021; Wang et al., 2023b), forward invariance (Ames et al., 2016; Taylor et al., 2020; Zhao et al., 2021; Wang et al., 2023a; Huang et al., 2023), stability (Lyapunov, 1992; Chang et al., 2019; Dai et al., 2021; Wu et al., 2023; Yang et al., 2024), etc.

In particular, we focus on the Lyapunov (Lyapunov, 1992) asymptotic stability of NN-based controllers in discrete-time nonlinear dynamical systems (Wu et al., 2023; Yang et al., 2024), where we aim to train and verify asymptotically Lyapunov-stable NN-based controllers. It involves training a controller while also finding a Lyapunov function which intuitively characterizes the energy of input states in the dynamical system, where the global minima of the Lyapunov function is at an equilibrium state. If it can be guaranteed that for any state within a region-of-attraction (ROA), the controller always makes the system evolve towards states with lower Lyapunov function values, then it implies that starting from any state within the ROA, the controller can always make the system converge towards the equilibrium state and thus the stability can be guaranteed. Such stability requirements have been formulated as the Lyapunov condition in the literature.

Previous works (Wu et al., 2023; Yang et al., 2024) typically used a counterexample-guided procedure that basically tries to find concrete inputs which violate the Lyapunov condition and then train models on counterexamples. After the training, the Lyapunov condition is verified by a formal verifier for NNs (Zhang et al., 2018; Xu et al., 2020; 2021; Wang et al., 2021; Zhang et al., 2022; Shi et al., 2024). However, the training process has very limited consideration on the computation of verification which is typically achieved by computing verified output bounds given an input region.

Thereby, their models are often not sufficiently "verification-friendly", and the verification can be challenging and take a long time after training (Yang et al., 2024).

In this paper, we propose to consider the computation of verification during the training, for the first time on the problem of learning Lyapunov-stable neural controllers. To do this, we optimize for *verified bounds on subregions of inputs* instead of only violations on concrete counterexample data points, and thus our approach differs significantly compared to Wu et al. (2023); Yang et al. (2024). Optimizing for verified bounds during training is also known as "certified training" which was originally proposed for training provably robust NNs (Wong & Kolter, 2018; Mirman et al., 2018; Gowal et al., 2018; Müller et al., 2022; Shi et al., 2021; De Palma et al., 2022; Mao et al., 2024) under adversarial robustness settings (Szegedy et al., 2014; Goodfellow et al., 2015). However, our certified training here is significantly different, as we require that the model should globally satisfy desired properties on an entire large region-of-interest over the input space, rather than only local robustness guarantees around a finite number of data points. Additionally, the model in this problem contains not only an NN as the controller, but also a Lyapunov function and nonlinear operators from the system dynamics, introducing additional difficulty to the training and verification.

We propose a new **C**ertified **T**raining framework enhanced with training-time **B**ranch-**a**nd-**B**ound, namely **CT-BaB**. We jointly train a NN controller and a Lyapunov function by computing and optimizing for the verified bounds on the violation of the Lyapunov condition. To achieve certified guarantees on the entire region-of-interest, we dynamically maintain a training dataset which consists of subregions in the region-of-interest. We split hard examples of subregions in the dataset into smaller ones during the training, along the input dimension where a split can yield the best improvement on the training objective, so that the training can be eased with tighter verified bounds for the smaller new subregions. Our new certified training framework is generally formulated for problems requiring guarantees on an entire input region-of-interest, but we focus on the particular problem of learning Lyapunov-stable controllers in this paper as a case study.

Our work makes the following contributions:

- We propose a new certified training framework for producing NNs with relatively global guarantees which provably hold on the entire input region-of-interest instead of only small local regions around a finite number of data points. We resolve challenges in certified training for the relatively large input region-of-interest by proposing a training-time branch-and-bound method with a dynamically maintained training dataset.
- We demonstrate the new certified training framework on the problem of learning (asymptotically) Lyapunov-stable neural controller. To the best of our knowledge, this is also the first certified training work for the task. Our new approach greatly reduced the training challenges observed in previous work. For example, unlike previous works (Chang et al., 2019; Wu et al., 2023; Yang et al., 2024) which required a special initialization from a linear quadratic regulator (LQR) during counterexample-guided training, our certified training approach works well by training from scratch with random initialization.
- We empirically show that our training framework produces neural controllers which verifiably satisfy the Lyapunov condition, with a larger region-of-attraction (ROA), and the Lyapunov condition can be much more efficiently verified at test time. On the largest 2D quadrotor dynamical system, we reduce the verification time from 1.1 hours (Yang et al., 2024) to 11.5 minutes, while our ROA size is 16X larger.

2 Related Work

Learning Lyapunov-stable neural controllers. On the problem of learning (asymptotically) Lyapunov-stable neural controllers, compared to methods using linear quadratic regulator (LQR) or sum-of-squares (SOS) (Parrilo, 2000; Tedrake et al., 2010; Majumdar et al., 2013; Yang et al., 2023; Dai & Permenter, 2023) to synthesize linear or polynomial controllers with Lyapunov stability guarantees (Lyapunov, 1992), NN-based controllers have recently shown great potential in scaling to more complicated systems with larger region-of-attraction. Some works used sampled data points to synthesize empirically stable neural controllers (Jin et al., 2020; Sun & Wu, 2021; Dawson et al., 2022; Liu et al., 2023) but they did not provide formal guarantees. Among them, although Jin et al. (2020) theoretically considered verification, they assumed an existence of some Lipschitz constant

which was not actually computed, and they only evaluated a finite number of data points without a formal verification.

To learn neural controllers with formal guarantees, many previous works used a Counter Example Guided Inductive Synthesis (CEGIS) framework by iteratively searching for counterexamples which violate the Lyapunov condition and then optimizing their models using the counterexamples, where counterexamples are generated by Satisfiable Modulo Theories (SMT) solvers (Gao et al., 2013; De Moura & Bjørner, 2008; Chang et al., 2019; Abate et al., 2020), Mixed Integer Programming solvers (Dai et al., 2021; Chen et al., 2021; Wu et al., 2023), or projected gradient descent (PGD) (Madry et al., 2018; Wu et al., 2023; Yang et al., 2024). Among these works, Wu et al. (2023) has also leveraged a formal verifier (Xu et al., 2020) only to guarantee that the Lyapunov function is positive definite (which can also be achieved by construction as done in Yang et al. (2024)) but not other more challenging parts of the Lyapunov condition; Yang et al. (2024) used α,β-CROWN (Zhang et al., 2018; Xu et al., 2020; 2021; Wang et al., 2021; Zhang et al., 2022; Shi et al., 2024) to verify trained models without using verified bounds for training. In contrast to those previous works, we propose to conduct certified training by optimizing for differentiable verified bounds at training time, where the verified bounds are computed for input subregions rather than violations on individual counterexample points, to produce more verification-friendly models.

Verification for neural controllers on other safety properties. Apart from Lyapunov asymptotic stability, there are many previous works on verifying other safety properties of neural controllers. Many works studied the reachability of neural controllers to verify the reachable sets of neural controllers and avoid reaching unsafe states (Althoff & Kochdumper, 2016; Dutta et al., 2019; Tran et al., 2020; Hu et al., 2020; Everett et al., 2021; Ivanov et al., 2021; Huang et al., 2022; Wang et al., 2023b; Schilling et al., 2022; Kochdumper et al., 2023; Jafarpour et al., 2023; 2024; Teuber et al., 2024). Additionally, many other works studied the forward invariance and barrier functions of neural controllers (Zhao et al., 2021; Wang et al., 2023a; Huang et al., 2023; Harapanahalli & Coogan, 2024; Hu et al., 2024; Wang et al., 2024). In contrast to the safety properties studied in those works, the Lyapunov asymptotic stability we study is a stronger guarantee which implies a convergence towards an equilibrium point, which is not guaranteed by reachability or forward invariance alone.

NN verification and certified training. On the general problem of verifying NN-based models on various properties, many techniques and tools have been developed in recent years, such as α,β-CROWN (Zhang et al., 2018; Xu et al., 2020; 2021; Wang et al., 2021; Zhang et al., 2022; Shi et al., 2024), nnenum (Bak, 2021), NNV (Tran et al., 2020; Lopez et al., 2023), MN-BaB (Ferrari et al., 2021), Marabou (Wu et al., 2024), NeuralSAT (Duong et al., 2024), VeriNet (Henriksen & Lomuscio, 2020), etc. One technique commonly used in the existing NN verifiers is linear relaxation-based bound propagation (Zhang et al., 2018; Wong & Kolter, 2018; Singh et al., 2019), which essentially relaxes nonlinear operators in the model by linear lower and upper bounds and then propagates linear bounds through the model to eventually produce a verified bound on the output of the model. Verified bounds computed in this way are differentiable and thus have also been leveraged in certified training (Zhang et al., 2020; Xu et al., 2020). Some other certified training works (Gowal et al., 2018; Mirman et al., 2018; Shi et al., 2021; Müller et al., 2022; De Palma et al., 2022) used even cheaper verified bounds computed by Interval Bound Propagation (IBP) which only propagates more simple interval bounds rather than linear bounds. However, existing certified training works commonly focused on adversarial robustness for individual data points with small local perturbations. In contrast, we consider a certified training beyond adversarial robustness, where we aim to achieve a relatively global guarantee which provably holds within the entire input region-of-interest rather than only around a proportion of individual examples.

Moreover, since verified bounds computed with linear relaxation can often be loose, many of the aforementioned verifiers for trained models also contain a branch-and-bound strategy (Bunel et al., 2020; Wang et al., 2021) to branch the original verification problem into subproblems with smaller input or intermediate bounds, so that the verifier can more tightly bound the output. In this work, we explore a novel use of the branch-and-bound concept in certified training, by dynamically expanding a training dataset and gradually splitting hard examples into smaller ones during the training, to enable certified training which eventually works for the entire input region-of-interest.

3 METHODOLOGY

3.1 PROBLEM SETTINGS

Certified training problem. Suppose the input region-of-interest of the problem is defined by $\mathcal{B} \subseteq \mathbb{R}^d$ for input dimension d, and in particular, we assume $\mathcal{B}$ is an axis-aligned bounding box $\mathcal{B} = \{\mathbf{x} \mid \underline{\mathbf{b}} \leq \mathbf{x} \leq \overline{\mathbf{b}}, \mathbf{x} \in \mathbb{R}^d\}$ with boundary defined by $\underline{\mathbf{b}}, \overline{\mathbf{b}} \in \mathbb{R}^d$ (we use "$\leq$" for vectors to denote that the "$\leq$" relation holds for all the dimensions in the vectors). We define a model (or a computational graph) $g_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}$ parameterized by $\boldsymbol{\theta}$, where $g_{\boldsymbol{\theta}}$ generally consists of one or more NNs and also additional operators which define the properties we want to certify (such as the Lyapunov condition in this work). The goal of certified training is to optimize for parameters $\boldsymbol{\theta}$ such that the following can be provably verified (we may omit $\boldsymbol{\theta}$ in the remaining part of the paper):

$$\forall \mathbf{x} \in \mathcal{B},\ g_{\boldsymbol{\theta}}(\mathbf{x}) \leq 0, \tag{1}$$

where any $g_{\boldsymbol{\theta}}(\mathbf{x}) > 0$ can be viewed as a violation. Unlike previous certified training works (Gowal et al., 2018; Mirman et al., 2018; Zhang et al., 2020; Müller et al., 2022) which only considered certified adversarial robustness guarantees on small local regions as $\{\mathbf{x} : \|\mathbf{x} - \mathbf{x}_0\| \leq \epsilon\}$ around a finite number of examples $\mathbf{x}_0 \in \mathcal{B}$ in the dataset, we require Eq. (1) to be fully certified for any $\mathbf{x} \in \mathcal{B}$.

Neural network verifiers typically verify Eq. (1) by computing a provable upper bound $\overline{g}$ such that $\overline{g} \geq g(\mathbf{x})$ ($\forall \mathbf{x} \in \mathcal{B}$) provably holds, and Eq. (1) is considered as verified if $\overline{g} \leq 0$. For models trained without certified training, the upper bound computed by verifiers is usually loose, or it requires a significant amount of time to further optimize the bounds or gradually tighten the bounds by branch-and-bound at test time. Certified training essentially optimizes for objectives which take the computation of verified bounds into consideration, so that Eq. (1) not only empirically holds for any worst-case data point $\mathbf{x}$ which can be empirically found to maximize $g(\mathbf{x})$, but also the model becomes more verification-friendly, i.e., verified bounds become tighter and thereby it is easier to verify $\overline{g} \leq 0$ with less branch-and-bound at test time.

Specifications for Lyapunov-stable neural control. In this work, we particularly focus on the problem of learning a certifiably Lyapunov-stable neural state-feedback controller with continuous control actions in a nonlinear discrete-time dynamical system, with asymptotic stability guarantees. We adopt the formulation from Yang et al. (2024). Essentially, there is a nonlinear dynamical system

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, u_t(\mathbf{x}_t)), \tag{2}$$

which takes the state $\mathbf{x}_t \in \mathbb{R}^d$ at the current time step t and a continuous control input $u_t(\mathbf{x}_t) \in \mathbb{R}^{n_u}$, and then the dynamical system determines the state at the next time step $t+1$. The control input $u_t(\mathbf{x}_t)$ is generated by a controller which is a NN here. The state of the dynamical system is also the input of the certified training problem.

Lyapunov asymptotic stability can guarantee that if the system begins at any state $\mathbf{x} \in \mathcal{S}$ within a region-of-attraction (ROA) $\mathcal{S} \subseteq \mathcal{B}$, it will converge to a stable equilibrium state $\mathbf{x}^*$. Following previous works, we assume that the equilibrium state is known, which can be manually derived from the system dynamics for the systems we study. To certify the Lyapunov asymptotic stability, we need to learn a Lyapunov function $V(\mathbf{x}_t) : \mathbb{R}^d \to \mathbb{R}$, such that the Lyapunov condition provably holds for the dynamical system in Eq. (2):

$$\forall \mathbf{x}_t \neq \mathbf{x}^* \in \mathcal{S},\ \ V(\mathbf{x}_t) > 0,\ \ V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t) \leq -\kappa V(\mathbf{x}_t), \tag{3}$$

and $V(\mathbf{x}^*) = 0$, where $\kappa > 0$ is a constant which specifies the exponential stability convergence rate. This condition essentially guarantees that at each time step, the controller always make the system progress towards the next state with a lower Lyapunov function value, and thereby the system will ultimately reach $\mathbf{x}^*$ which has the lowest Lyapunov function value given $V(\mathbf{x}^*) = 0$. Following Yang et al. (2024), we guarantee $V(\mathbf{x}^*) = 0$ and $\forall \mathbf{x}_t \neq \mathbf{x} \in \mathbb{R}^d, V(\mathbf{x}_t) > 0$ by the construction of the Lyapunov function, as discussed in Section 3.4, and we specify the ROA using a sublevel set of V as $\mathcal{S} := \{\mathbf{x} \in \mathcal{B} \mid V(\mathbf{x}) < \rho\}$ with sublevel set threshold ρ. Since ROA is now restricted to be a subset of $\mathcal{B}$ and the verification will only focus on $\mathcal{B}$, we additionally need to ensure that the state at the next time step does not leave $\mathcal{B}$, i.e., $\mathbf{x}_{t+1} \in \mathcal{B}$.

Overall, we want to verify $g(\mathbf{x}_t) \leq 0$ for all $\mathbf{x}_t \in \mathcal{B}$, where $g(\mathbf{x}_t)$ is defined as:

$$g(\mathbf{x}_t) := \min \left\{ \rho - V(\mathbf{x}_t),\ \sigma(V(\mathbf{x}_{t+1}) - (1-\kappa)V(\mathbf{x}_t)) + \sum_{1 \leq i \leq d} \sigma([\mathbf{x}_{t+1}]_i - \overline{\mathbf{b}}_i) + \sigma(\underline{\mathbf{b}}_i - [\mathbf{x}_{t+1}]_i) \right\}, \tag{4}$$

where $\mathbf{x}_{t+1}$ is given by Eq. (2), and $\sigma(x) = \max\{x, 0\}$ is also known as ReLU. For the specification in Eq. (4), $\rho - V(\mathbf{x}_t)$ means that for a state which is provably out of the considered ROA as $V(\mathbf{x}_t) \geq \rho$, we do not have to verify Eq. (3) or $\mathbf{x}_{t+1} \in \mathcal{B}$, and it immediately satisfies $g(\mathbf{x}_t) \leq 0$; $\sigma(V(\mathbf{x}_{t+1}) - (1-\kappa)V(\mathbf{x}_t))$ is the violation on the $V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t) \leq -\kappa V(\mathbf{x}_t)$ condition in Eq. (3); and the "$\sum_{1 \leq i \leq d}$" term in Eq. (4) denotes the violation on the $\mathbf{x}_{t+1} \in \mathcal{B}$ condition. Verifying Eq. (4) for all $\mathbf{x}_t \in \mathcal{B}$ guarantees the Lyapunov condition for any $\mathbf{x} \in \mathcal{S}$ in the ROA (Yang et al., 2024). In the training, we try to make $g(\mathbf{x}_t) \leq 0$ verifiable by optimizing the parameters in the neural controller u_t and the Lyapunov function $V(\mathbf{x}_t)$.

3.2 Training Framework

As we are now considering a challenging setting, where we want to guarantee $g(\mathbf{x}) \leq 0$ on the entire input region-of-interest $\mathcal{B}$, directly computing a verified bound on the entire $\mathcal{B}$ can produce very loose bounds. Thus, we split $\mathcal{B}$ into smaller subregions, and we we maintain a dataset with n examples $\mathbb{D} = \{(\underline{\mathbf{x}}^{(1)}, \overline{\mathbf{x}}^{(1)}), (\underline{\mathbf{x}}^{(2)}, \overline{\mathbf{x}}^{(2)}), \cdots, (\underline{\mathbf{x}}^{(n)}, \overline{\mathbf{x}}^{(n)})\}$, where each example $(\underline{\mathbf{x}}^{(k)}, \overline{\mathbf{x}}^{(k)})$ $(1 \leq k \leq n)$ is a subregion in $\mathcal{B}$, defined as a bounding box $\{\mathbf{x} : \mathbf{x} \in \mathbb{R}^d,\ \underline{\mathbf{x}}^{(k)} \leq \mathbf{x} \leq \overline{\mathbf{x}}^{(k)}\}$ with boundary $\underline{\mathbf{x}}^{(k)}$ and $\overline{\mathbf{x}}^{(k)}$, and all the examples in $\mathbb{D}$ cover $\mathcal{B}$ as $\bigcup_{(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \in \mathbb{D}} (\underline{\mathbf{x}}, \overline{\mathbf{x}}) = \mathcal{B}$. We dynamically update and expand the dataset during the training by splitting hard examples into more examples with even smaller subregions, as we will introduce in Section 3.3.

During the training, for each training example $(\underline{\mathbf{x}}, \overline{\mathbf{x}})$, we compute a verified upper bound of $g(\mathbf{x})$ for all $\mathbf{x}$ $(\underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}})$ within the subregion, denoted as $\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}})$, such that

$$\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \geq g(\mathbf{x}) \ (\forall \mathbf{x},\ \underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}}). \tag{5}$$

Thereby, $\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}})$ is a verifiable upper bound on the worst-case violation of Eq. (1) for data points in $[\underline{\mathbf{x}}, \overline{\mathbf{x}}]$. To compute $\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}})$, we mainly use the CROWN (Zhang et al., 2018; 2020) algorithm which is based on linear relaxation-based bound propagation as mentioned in Section 2, while we also use a more simple Interval Bound Propagation (IBP) (Gowal et al., 2018; Mirman et al., 2018) algorithm to compute the intermediate bounds of the hidden layers in NNs. Such intermediate bounds are required by CROWN to derive linear relaxation for nonlinear operators including activation functions, as well as nonlinear computation in the dynamics of the dynamical system. We use IBP on hidden layers for more efficient training and potentially easier optimization (Lee et al., 2021; Jovanović et al., 2021). Verified bounds computed in this way are differentiable, and then we aim to achieve $\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \leq 0$ and minimize $\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}})$ in the training.

We additionally include a training objective term where we try to empirically find the worst-case violation of Eq. (1) by adversarial attack using projected gradient descent (PGD) (Madry et al., 2018), denoted as $\overline{g}^A(\underline{\mathbf{x}}, \overline{\mathbf{x}}) := g(A(\underline{\mathbf{x}}, \overline{\mathbf{x}}))$, where $A(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \in \mathbb{R}^d$ $(\underline{\mathbf{x}} \leq A(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \leq \overline{\mathbf{x}})$ is a data point found by PGD to empirically maximize $g(A(\underline{\mathbf{x}}, \overline{\mathbf{x}}))$ within the domain:

$$\underset{\mathbf{x} \in \mathbb{R}^d\ (\underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}})}{\arg\max} \quad g(\mathbf{x}), \tag{6}$$

but $A(\underline{\mathbf{x}}, \overline{\mathbf{x}})$ found by PGD is not guaranteed to be the optimal solution for Eq. (6). Since it is easier to train a model which empirically satisfies Eq. (1) compared to making Eq. (1) verifiable, we add this adversarial attack objective so that the training can more quickly reach a solution with most counterexamples eliminated, while certified training can focus on making it verifiable. This objective also helps to achieve that at least no counterexample can be empirically found, even if verified bounds by CROWN and IBP cannot yet verify all the examples in the current dataset $(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \in \mathbb{D}$, as we may still be able to fully verify Eq. (1) at test time using a stronger verifier enhanced with large-scale branch-and-bound.

Overall, we optimize for a loss function to minimize the violation of $\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}})$ and $\overline{g}^A(\underline{\mathbf{x}}, \overline{\mathbf{x}})$:

$$L = \left(\mathbb{E}_{(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \in \mathbb{D}}\ \sigma(\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}}) + \epsilon) + \lambda \max \sigma(\overline{g}^A(\underline{\mathbf{x}}, \overline{\mathbf{x}}) + \epsilon) \right) + L_{\text{extra}}, \tag{7}$$

where $\sigma(\cdot)$ is ReLU, ϵ is small value for ideally achieving Eq. (1) with a margin, as $\overline{g}(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \leq -\epsilon$ and $\overline{g}^A(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \leq -\epsilon$, λ is a coefficient used to for assigning a weight to the PGD term, and L_{extra} is an extra loss term which can be used to control additional properties of the model. After the training, the desired properties as Eq. (1) are verified by a formal verifier such as α,β-CROWNwith larger-scale branch-and-bound, and thus the soundness of the trained models can be guaranteed as long as the verification succeeds at test time.

We have formulated our general training framework in this section, and we will instantiate our training framework on the particular task of learning Lyapunov-stable neural controllers in Section 3.4.

3.3 Training-Time Branch-and-Bound

We now discuss how we initialize the training dataset $\mathbb{D}$ and dynamically maintain the dataset during the training by splitting hard examples into smaller subregions.

Initial splits. We initialize $\mathbb{D}$ by splitting the original input region-of-interest $\mathcal{B}$ into grids along each of its d dimensions, respectively. We control the maximum size of the initial regions with a threshold l which denotes the maximum length of each input dimension. For each input dimension i $(1 \leq i \leq d)$, we uniformly split the input range $[\underline{\mathbf{b}}_i, \overline{\mathbf{b}}_i]$ into $m_i = \lceil \frac{\overline{\mathbf{b}}_i - \underline{\mathbf{b}}_i}{l} \rceil$ segments in the initial split, such that the length of each segment is no larger than the threshold l. We thereby create $\prod_{i=1}^{d} m_i$ regions to initialize $\mathbb{D}$, where each region is created by taking a segment from each input dimension, respectively. Each region $(\underline{\mathbf{x}}, \overline{\mathbf{x}}) \in \mathbb{D}$ is also an example in the training dataset. We set the threshold l such that the initial examples fill 1$\sim$2 batches according to the batch size, so that the batch size can remain stable in the beginning of the training rather than start with a small actual batch size.

Splits during the training. After we create the initial splits with uniform splits along each input dimension, during the training, we also dynamically split hard regions into even smaller subregions. We take dynamic splits instead of simply taking more initial splits, as we can leverage the useful information during the training to identify hard examples to split where the specification has not been verified, and we also identify the input dimension to split such that it can lead to the best improvement on the loss values.

In each training batch, we take each example $(\underline{\mathbf{x}}^{(k)}, \overline{\mathbf{x}}^{(k)})$ with $\overline{g}(\underline{\mathbf{x}}^{(k)}, \overline{\mathbf{x}}^{(k)}) > 0$, i.e., we have not been able to verify that $g(\mathbf{x}) \leq 0$ within the region $[\underline{\mathbf{x}}^{(k)}, \overline{\mathbf{x}}^{(k)}]$. We then choose one of the input dimensions $i(1 \leq i \leq d)$ and uniformly split the region into two subregions along the chosen input dimension i. At dimension i, suppose the original input range for the example is $[\underline{\mathbf{x}}_i^{(k)}, \overline{\mathbf{x}}_i^{(k)}]$, we split it into $[\underline{\mathbf{x}}_i^{(k)}, \frac{\underline{\mathbf{x}}_i^{(k)} + \overline{\mathbf{x}}_i^{(k)}}{2}]$ and $[\frac{\underline{\mathbf{x}}_i^{(k)} + \overline{\mathbf{x}}_i^{(k)}}{2}, \overline{\mathbf{x}}_i^{(k)}]$, while leaving other input dimensions unchanged. We remove the original example from the dataset and add the two new subregions into the dataset.

In order to maximize the benefit of splitting an example, we decide the input dimension to choose by trying each of the input dimensions $j(1 \leq j \leq d)$ and computing the total loss of the two new subregions when dimension j is split. Suppose $L(\underline{\mathbf{x}}^{(k)}, \overline{\mathbf{x}}^{(k)})$ is the contribution of an example $(\underline{\mathbf{x}}^{(k)}, \overline{\mathbf{x}}^{(k)})$ to the loss function in Eq. (7). We take the dimension j to split which leads to the lowest loss value for the new examples:

$$\arg\min_{1 \leq j \leq d} \; L(\underline{\mathbf{x}}^{(k)}, \overline{\mathbf{x}}^{(k,j)}) + L(\underline{\mathbf{x}}^{(k,j)}, \overline{\mathbf{x}}^{(k)}), \quad \text{where } \underline{\mathbf{x}}_j^{(k,j)} = \overline{\mathbf{x}}_j^{(k,j)} = \frac{\underline{\mathbf{x}}_j^{(k)} + \overline{\mathbf{x}}_j^{(k)}}{2}, \tag{8}$$

and $\underline{\mathbf{x}}_i^{(k,j)} = \underline{\mathbf{x}}_i^{(k)}, \overline{\mathbf{x}}_i^{(k,j)} = \overline{\mathbf{x}}_i^{(k)}$ keep unchanged for other dimensions $i \neq j$ not being split. All the examples requiring a split in a batch and all the input dimensions to consider for the split can be handled in parallel on GPU.

3.4 Modeling and Training Objectives for Lyapunov-stable Neural Control

To demonstrate our new certified training framework, we focus on its application on learning verifiably Lyapunov-stable neural controllers with state feedback. Since our focus is on a new certified

training framework, we use the same model architecture as Yang et al. (2024). We use a fully-connected NN for the controller $u(\mathbf{x})$; for the Lyapunov function $V(\mathbf{x})$, we either use a model based on a fully-connected NN $\phi(\mathbf{x})$ as $V(\mathbf{x}) = |\phi(\mathbf{x}) - \phi(\mathbf{x}^*)| + \|(\epsilon_V I + R^\top R)(\mathbf{x} - \mathbf{x}^*)\|_1$, or a quadratic function as $V(\mathbf{x}) = (\mathbf{x} - \mathbf{x}^*)^\top(\epsilon_V I + R^\top R)(\mathbf{x} - \mathbf{x}^*)$, where $R \in \mathbb{R}^{n_r \times n_r}$ is an optimizable matrix parameter, and $\epsilon_V > 0$ is a small positive value to guarantee that $\epsilon_V I + R^\top R$ is positive definite. The construction of the Lyapunov functions automatically guarantees that $V(\mathbf{x}^*) = 0$ and $V(\mathbf{x}) > 0$ $(\forall \mathbf{x} \neq \mathbf{x}^*)$ (Yang et al., 2024) required in the Lyapunov condition.

We have discussed the formulation of $g(\mathbf{x})$ in Eq. (4). When bounding the violation term $V(\mathbf{x}_{t+1}) - (1-\kappa)V(\mathbf{x}_t)$ in Eq. (4), we additionally apply a constraint $V(\mathbf{x}_{t+1}) \geq \rho + \epsilon$ for $\mathbf{x}_{t+1} \notin \mathcal{B}$. It is to prevent wrongly minimizing the violation by going out of the region-of-interest as $\mathbf{x}_{t+1} \notin \mathcal{B}$ while making $V(\mathbf{x}_{t+1})$ $(\mathbf{x}_{t+1} \notin \mathcal{B})$ small, such that the violation $V(\mathbf{x}_{t+1}) - (1-\kappa)V(\mathbf{x}_t)$ appears to be small yet missing the $\mathbf{x}_{t+1} \in \mathcal{B}$ requirement.

As mentioned in Eq. (4), an additional term L_{extra} can be added to control additional properties of the model. We use the extra loss term to control the size of the region-of-attraction (ROA). We aim to have a good proportion of data points from the region-of-interest $\mathbf{x} \in \mathcal{B}$, such that their Lyapunov function values are within the sublevel set $V(\mathbf{x}) < \rho$ where the Lyapunov condition is to be guaranteed. To do this, we randomly draw a batch of n_ρ samples within $\mathcal{B}$, as $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \cdots, \tilde{\mathbf{x}}_{n_\rho} \in \mathcal{B}$, and we define L_{extra} as:

$$L_{\text{extra}} = \mathbb{I}\left(\frac{1}{n_\rho}\sum_{i=1}^{n_\rho}\mathbb{I}(V(\tilde{\mathbf{x}}_i) < \rho) < \rho_{\text{ratio}}\right)\frac{\lambda_\rho}{n_\rho}\sum_{i=1}^{n_\rho}\sigma(V(\tilde{\mathbf{x}}_i) + \rho - \epsilon), \tag{9}$$

which penalizes samples with $V(\tilde{\mathbf{x}}_i) > \rho - \epsilon$ when the ratio of samples within the sublevel set is below the threshold ρ_{ratio}, where ϵ is a small value for the margin as similarly used in Eq. (7) and λ_ρ is the weight of term L_{extra} Eq. (7). In our implementation, we simply fix $\rho = 1$ and make n_ρ equal to the batch size of the training. The threshold ρ_{ratio} and the weight λ_ρ can be set to reach the desired ROA size, but setting a stricter requirement on the ROA size can naturally increase the difficulty of training.

All of our models are randomly initialized and trained from scratch. This provides an additional benefit compared to previous works (Wu et al., 2023; Yang et al., 2024) which commonly required an initialization for a traditional non-learning method (linear quadratic regulartor, LQR) with a small ROA. Yang et al. (2024) also proposed to enlarge ROA with carefully selected candidates states which are desired to be within the ROA by referring to LQR solutions. In contrast, our training does not require any baseline solution. Thus, this improvement from our method can reduce the burden of applying our method without requiring a special initialization.

4 Experiments

4.1 Experimental Settings

Table 1: Dynamical systems used in the experiments. All these settings follow Yang et al. (2024). d means the dimension of input states and n_u means the dimension of control input which is from the output of the controller. There is a limit on the control input u and the output of the controller is clamped according to the limit, where some symbols in the limit on u are from the dynamics of the systems: m for mass, g for gravity, l for length, and v for velocity. Size of the region-of-interest here is represented by the upper boundary $\overline{\mathbf{b}}$, and $\underline{\mathbf{b}} = -\overline{\mathbf{b}}$ holds for all the systems here. Equilibrium state of all the systems here is $\mathbf{x}^* = \mathbf{0}$.

System	d	n_u	Limit on u	Region-of-interest
Inverted pendulum	2	1	$\lvert u\rvert \leq 8.15 \cdot mgl$ (large torque) $\lvert u\rvert \leq 1.02 \cdot mgl$ (small torque)	$[12, 12]$
Path tracking	2	1	$\lvert u\rvert \leq 1.68 \cdot l/v$ (large torque) $\lvert u\rvert \leq l/v$ (small torque)	$[3, 3]$
2D quadrotor	6	3	$\lVert u\rVert_\infty \leq 1.25 \cdot mg$	$[0.75, 0.75, \pi, 2, 4, 4, 3]$

Table 2: Comparison on the verification time cost and the size of ROA. "Pendulum" refers to the inverted pendulum system. Model checkpoints for Wu et al. (2023) are obtained from the source code of Yang et al. (2024) and the same models have been used for comparison in Yang et al. (2024), where "-" denotes that on some of the systems models for Wu et al. (2023) are not available. Yang et al. (2024) and ours have the same model architecture on each system.

System	Wu et al. (2023)		CEGIS (Yang et al., 2024)		Ours	
	Time	ROA	Time	ROA	Time	ROA
Pendulum (large torque)	11.3s	53.28	33s	239.04	32s	**495.36**
Pendulum (small torque)	-	-	25s	187.20	26s	**275.04**
Path tracking (large torque)	11.7s	14.38	39s	18.27	**31s**	**21.60**
Path tracking (small torque)	-	-	34s	10.53	**27s**	**11.51**
2D quadrotor	-	-	1.1hrs	3.29	**11.5min**	**54.39**

Table 3: Runtime of training, size of the training dataset, and the ratio of examples in the training dataset verifiable by CROWN without further branch-and-bound. "Initial dataset size" denotes the size of the training dataset at the start of the training, and "final dataset size" denote the size at the end of the training. All the models can be fully verified at test time using α,β-CROWN with branch-and-bound at the input space, as shown in Table 2.

System	Runtime	Initial dataset size	Final dataset size	Verified by CROWN
Pendulum (large torque)	6min	58080	68686	100%
Pendulum (small torque)	32min	58080	657043	100%
Path tracking (large torque)	17min	40400	7586381	94.95%
Path tracking (small torque)	16min	40400	222831	99.97%
2D quadrotor	107min	46336	34092930	88.18%

Dynamical systems. We demonstrate our new certified training work on learning Lyapunov-stable neural controllers with state feedback in several nonlinear discrete-time dynamical systems following Wu et al. (2023); Yang et al. (2024), as listed in Table 1: *Inverted pendulum* is about swinging up the pendulum to the upright equilibrium; *Path tracking* is about tracking a path for a planar vehicle; and *2D quadrotor* is about hovering a quadrotor at the equilibrium state. For inverted pendulum and path tracking, there are two different limits on the maximum allowed torque of the controller, where the setting is more challenging with a smaller torque limit. Detailed definition of the system dynamics (f in Eq. (2)) is available in existing works: Wu et al. (2023) for inverted pendulum and path tracking, and Tedrake (2009) for 2D quadrotor.

Implementation. We use the PyTorch library auto_LiRPA (Xu et al., 2020) to compute CROWN and IBP verified bounds during the training. After a model is trained, we use α,β-CROWN to finally verify the trained model, where α,β-CROWN is configured to use verified bounds by auto_LiRPA and run branch-and-bound on the input space to tighten the verified bounds until the verification succeeds, which has been used in the same way in Yang et al. (2024). Additional details of the experiments are included in Appendix A.

4.2 Main Results

We show the main results in Table 2, where we compare the verification time cost and size of ROA with the previous state-of-the-art method based on CEGIS (Yang et al., 2024), as well as an earlier work (Wu et al., 2023) on applicable systems. Following Wu et al. (2023), we estimate the size of ROA by considering grid points in the region-of-interest $\mathcal{B}$ and counting the proportion of grid points within the sublevel set of the Lyapunov function where the Lyapunov condition is verified, multiplied by the volume of $\mathcal{B}$. Models by Wu et al. (2023) have much smaller ROA than Yang et al. (2024), and thus we focus on comparing our method with Yang et al. (2024). On inverted pendulum, our method produces much larger ROA with similar verification time, and on path tracking, our method produces larger ROA while also reducing the verification time. On these two systems, the verification time cannot be greatly reduced, due to the overhead of launching α,β-CROWN and low GPU utilization

Table 4: Training and test results of ablation study conducted on the 2D quadrotor system. For training results, we report the dataset size at the end of the training and the ratio of training examples verified by CROWN, where "verified (all)" is evaluated on all the training examples, while "verified (within the sublevel set)" excludes examples verified to be out of the sublevel set with $V(\mathbf{x}) < \rho$. For test results, we report if the model can be fully verified at test time by α,β-CROWN and a "candidate ROA" size which denotes the volume of the sublevel set with $V(\mathbf{x}) < \rho$. "Candidate ROA" is the true ROA if the model is fully verified.

Method	Training			Test	
	Dataset size	Verified (all)	Verified (within the sublevel set)	Fully verified	Candidate ROA
Default	34092930	88.18%	**86.95%**	**Yes**	**54.39**
No dynamic split	64916160	99.95%	38.29%	No	0.08
Naive dynamic split	20477068	90.05%	55.62%	No	0.0095

when the verification is relatively easy. On 2D quadrotor with a much higher difficulty, our method significantly reduces the verification time (11.5 minutes compared to 1.1 hours by Yang et al. (2024)) while also significantly enlarging the ROA (54.39 compared to 3.29 by Yang et al. (2024)). These results demonstrate the effectiveness of our method on producing verification-friendly Lyapunov-stable neural controllers and Lyapunov functions with larger ROA. In Figure 1, we visualize the ROA on 2D quadrotor, with different 2D views, which demosntrates a larger ROA compared to the Yang et al. (2024) baseline. In Appendix B, we visualize the distribution of the subregions after our training-time branch-and-bound, which suggests that much more extensive splits tend to happen when at least one of the input states is close to that of the equilibrium state, where Lyapunov function values are relatively small and the training tends to be more challenging.

In Table 3, we show information about the training, including the time cost of training, size of the dynamic training dataset and the ratio of training examples which can be verified using verified bounds by CROWN (Zhang et al., 2018; Xu et al., 2020) at the end of the training. Our training dataset is dynamically maintained and expanded as described in Section 3.3, and the dataset size grows from the "initial dataset size" to the "final dataset size" shown in Table 3. At the end of the training, most of the training examples (more than 88%) can already be verified by CROWN bounds. Although not all of the training examples are verifiable by CROWN, all the models can be fully verified when we use α,β-CROWN to finally verify the models at test time, where α,β-CROWN further conducts branch-and-bound on the input space using CROWN bounds.

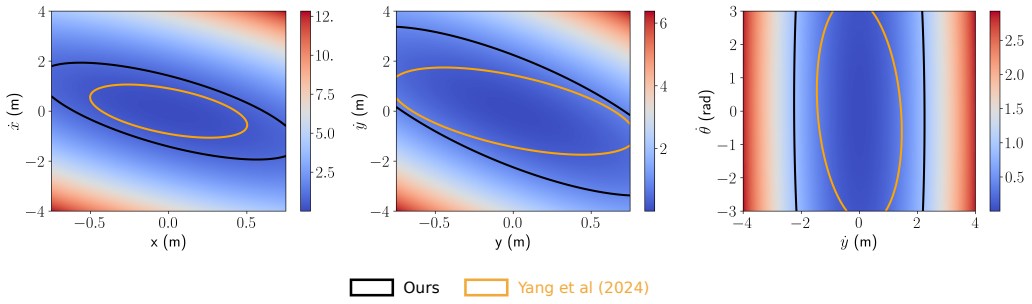


Figure 1: Visualization of the Lyapunov function (color plots) and ROA (contours) on the 2D quadrotor system with three different 2D views compared to Yang et al. (2024). The system contains 6 states denoted as $\mathbf{x} = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]$. Our method demonstrates a 16X larger ROA (in terms of the volume of ROA on the 6-dimensional input space) compared to Yang et al. (2024).

4.3 Ablation Study

In this section, we conduct an ablation study to demonstrate the necessity of using our dynamic splits to maintain the training dataset as described in Section 3.3, on the largest 2D quadrotor system. We consider two variations of our proposed method: *No dynamic split* means that we use a large number of initial splits by reducing the threshold l which controls the maximize size of initial regions

mentioned in Section 3.3, and the dataset is then fixed and there is no dynamic split throughout the training; *Naive dynamic split* means that we use dynamic splits but we simply split along the input dimension with the largest size, as $\arg\max_{1\leq j\leq d}(\overline{\mathbf{x}}_i^{(k)} - \underline{\mathbf{x}}_j^{(k)})$, instead of taking the best input dimension in terms of reducing the loss value as Eq. (8). We show the results in Table 4. Neither of "no dynamic split" and "naive dynamic split" can produce verifiable models. We observe that they both tend to make the sublevel set with $V(\mathbf{x}) < \rho$ very small, which leads to a very small ROA size even if the model can be verified (if the weight on the extra loss term for ROA in Eq. (9) is increased, the training does not converge with many counterexamples which can be empirically found). For the two variations, although most of the training examples can still be verified at the end of training, if we check nontrivial examples which are not verified to be out of the sublevel set (see "verified (within the sublevel set)" in Table 4), a much smaller proportion of these examples are verified. Without our proposed dynamic splits decided by Eq. (8), these two variations cannot identify hard examples to split and split along the best input dimension to efficiently ease the training, leaving many unverified examples among those possibly within the sublevel set, despite that the size of the sublevel set is significantly shrunk. This experiment demonstrates the benefit of our proposed dynamic splits.

5 Conclusion

To conclude, we propose a new certified training framework for training verification-friendly models where a relatively global guarantee can be verified for an entire region-of-interest in the input space. We maintain a dynamic dataset of subregions which cover the region-of-interest, and we split hard examples into smaller subregions throughout the training, to ease the training with tighter verified bounds. We demonstrate our new certified training framework on the problem of learning and verifying Lyapunov-stable neural controllers. We show that our new method produces more verification-friendly models which can be more efficiently verified at test time while the region-of-attraction also becomes much larger compared to the state-of-the-art baseline.

A limitation of this work is that only low-dimensional dynamical systems have been considered, which is also a common limitation of previous works on this Lyapunov problem (Chang et al., 2019; Wu et al., 2023; Yang et al., 2024). Future works may consider scaling up our method to higher-dimensional systems. Since splitting regions on the input space can become less efficient if the dimension of the input space significantly increases, future works may consider applying splits on the intermediate bounds of activation functions (potentially with sparsity), which has been commonly used in state-of-the-art NN verifiers (mentioned in Section 2) for verifying trained models on high-dimensional tasks such as image classification.

Although our new certified training framework is generally formulated, we have only focused on demonstrating the training framework on Lyapunov asymptotic stability. Given the generality of our new framework, it has the potential to enable broader applications, such as other safety properties including reachability and forward invariance mentioned in Section 2, control systems with more complicated settings such as output feedback systems, or even applications beyond control. These will be interesting directions for future work.

References

Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. Formal synthesis of lyapunov neural networks. *IEEE Control Systems Letters*, 5(3):773–778, 2020.

Matthias Althoff and Niklas Kochdumper. Cora 2016 manual. *TU Munich*, 85748, 2016.

Aaron D Ames, Xiangru Xu, Jessy W Grizzle, and Paulo Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8): 3861–3876, 2016.

Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pp. 19–36. Springer, 2021.

Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 2242–2253. IEEE, 2017.

Rudy Bunel, P Mudigonda, Ilker Turkaslan, P Torr, Jingyue Lu, and Pushmeet Kohli. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(2020), 2020.

Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

Shaoru Chen, Mahyar Fazlyab, Manfred Morari, George J Pappas, and Victor M Preciado. Learning lyapunov functions for hybrid systems. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pp. 1–11, 2021.

Hongkai Dai and Frank Permenter. Convex synthesis and verification of control-lyapunov and barrier functions with input constraints. In *IEEE American Control Conference (ACC)*, 2023.

Hongkai Dai, Benoit Landry, Lujie Yang, Marco Pavone, and Russ Tedrake. Lyapunov-stable neural-network control. *arXiv preprint arXiv:2109.14152*, 2021.

Charles Dawson, Zengyi Qin, Sicun Gao, and Chuchu Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, 2022.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008.

Alessandro De Palma, Rudy Bunel, Krishnamurthy Dvijotham, M Pawan Kumar, and Robert Stanforth. Ibp regularization for verified adversarial robustness via branch-and-bound. *arXiv preprint arXiv:2206.14772*, 2022.

Hai Duong, Dong Xu, ThanhVu Nguyen, and Matthew B Dwyer. Harnessing neuron stability to improve dnn verification. *Proceedings of the ACM on Software Engineering*, 1(FSE):859–881, 2024.

Souradeep Dutta, Xin Chen, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Sherlock-a tool for verification of neural network feedback systems: demo abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 262–263, 2019.

Michael Everett, Golnaz Habibi, Chuangchuang Sun, and Jonathan P How. Reachability analysis of neural feedback loops. *IEEE Access*, 2021.

Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. Complete verification via multi-neuron relaxation guided branch-and-bound. In *International Conference on Learning Representations*, 2021.

Sicun Gao, Soonho Kong, and Edmund M Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International conference on automated deduction*, pp. 208–214. Springer, 2013.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Akash Harapanahalli and Samuel Coogan. Certified robust invariant polytope training in neural controlled odes. *arXiv preprint arXiv:2408.01273*, 2024.

Patrick Henriksen and Alessio Lomuscio. Efficient neural network verification via adaptive refinement and adversarial search. In *ECAI 2020*, pp. 2513–2520. IOS Press, 2020.

Haimin Hu, Mahyar Fazlyab, Manfred Morari, and George J Pappas. Reach-sdp: Reachability analysis of closed-loop systems with neural network controllers via semidefinite programming. In *2020 59th IEEE conference on decision and control (CDC)*, pp. 5929–5934. IEEE, 2020.

Hanjiang Hu, Yujie Yang, Tianhao Wei, and Changliu Liu. Verification of neural control barrier functions with symbolic derivative bounds propagation. In *8th Annual Conference on Robot Learning*, 2024.

Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. Polar: A polynomial arithmetic framework for verifying neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 414–430. Springer, 2022.

Yujia Huang, Ivan Dario Jimenez Rodriguez, Huan Zhang, Yuanyuan Shi, and Yisong Yue. Fi-ode: Certified and robust forward invariance in neural odes. *arXiv*, 2023.

Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *International Conference on Computer Aided Verification*, pp. 249–262. Springer, 2021.

Saber Jafarpour, Akash Harapanahalli, and Samuel Coogan. Interval reachability of nonlinear dynamical systems with neural network controllers. In *Learning for Dynamics and Control Conference*, pp. 12–25. PMLR, 2023.

Saber Jafarpour, Akash Harapanahalli, and Samuel Coogan. Efficient interaction-aware interval analysis of neural network feedback loops. *IEEE Transactions on Automatic Control*, 2024.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Neural certificates for safe control policies. *arXiv preprint arXiv:2006.08465*, 2020.

Nikola Jovanović, Mislav Balunović, Maximilian Baader, and Martin Vechev. On the paradox of certified training. *arXiv preprint arXiv:2102.06700*, 2021.

Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open-and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods Symposium*, pp. 16–36. Springer, 2023.

Sungyoon Lee, Woojin Lee, Jinseong Park, and Jaewook Lee. Towards better understanding of training certifiably robust models against adversarial examples. *Advances in Neural Information Processing Systems*, 34:953–964, 2021.

Simin Liu, Changliu Liu, and John Dolan. Safe control under input limits with neural control barrier functions. In *Conference on Robot Learning*. PMLR, 2023.

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*, pp. 397–412. Springer, 2023.

Aleksandr Mikhailovich Lyapunov. The general problem of the stability of motion. *International journal of control*, 55(3):531–534, 1992.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Anirudha Majumdar, Amir Ali Ahmadi, and Russ Tedrake. Control design along trajectories with sums of squares programming. In *2013 IEEE International Conference on Robotics and Automation*, pp. 4054–4061. IEEE, 2013.

Yuhao Mao, Mark Müller, Marc Fischer, and Martin Vechev. Connecting certified and adversarial training. *Advances in Neural Information Processing Systems*, 36, 2024.

Matthew Mirman, Timon Gehr, and Martin T. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3575–3583, 2018.

Mark Niklas Müller, Franziska Eckert, Marc Fischer, and Martin Vechev. Certified training: Small boxes are all you need. *arXiv preprint arXiv:2210.04871*, 2022.

Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. California Institute of Technology, 2000.

Christian Schilling, Marcelo Forets, and Sebastián Guadalupe. Verification of neural-network control systems by integrating taylor models and zonotopes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8169–8177, 2022.

Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Fast certified robust training with short warmup. *Advances in Neural Information Processing Systems*, 34, 2021.

Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities. *arXiv preprint arXiv:2405.21063*, 2024.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41, 2019.

Wei Sun and Tianfu Wu. Learning layout and style reconfigurable gans for controllable image synthesis. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5070–5087, 2021.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

Andrew Taylor, Andrew Singletary, Yisong Yue, and Aaron Ames. Learning for safety-critical control with control barrier functions. In *Learning for Dynamics and Control*, pp. 708–717. PMLR, 2020.

Russ Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines. *Course notes for MIT*, 6:832, 2009.

Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 2010.

Samuel Teuber, Stefan Mitsch, and André Platzer. Provably safe neural network controllers via differential dynamic logic. *arXiv preprint arXiv:2402.10998*, 2024.

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*, 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.

Xinyu Wang, Luzia Knoedler, Frederik Baymler Mathiesen, and Javier Alonso-Mora. Simultaneous synthesis and verification of neural control barrier functions through branch-and-bound verification-in-the-loop training. In *2024 European Control Conference (ECC)*, pp. 571–578. IEEE, 2024.

Yixuan Wang, Simon Sinong Zhan, Ruochen Jiao, Zhilu Wang, Wanxin Jin, Zhuoran Yang, Zhaoran Wang, Chao Huang, and Qi Zhu. Enforcing hard constraints with soft barriers: Safe reinforcement learning in unknown stochastic environments. In *International Conference on Machine Learning*, pp. 36593–36604. PMLR, 2023a.

Yixuan Wang, Weichao Zhou, Jiameng Fan, Zhilu Wang, Jiajun Li, Xin Chen, Chao Huang, Wenchao Li, and Qi Zhu. Polar-express: Efficient and precise formal reachability analysis of neural-network controlled systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023b.

Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5283–5292, 2018.

Haoze Wu, Omri Isac, Aleksandar Zeljić, Teruhiro Tagomori, Matthew Daggitt, Wen Kokke, Idan Refaeli, Guy Amir, Kyle Julian, Shahaf Bassan, et al. Marabou 2.0: a versatile formal analyzer of neural networks. In *International Conference on Computer Aided Verification*, pp. 249–264. Springer, 2024.

Junlin Wu, Andrew Clark, Yiannis Kantaros, and Yevgeniy Vorobeychik. Neural lyapunov control for discrete-time systems. *Advances in neural information processing systems*, 36:2939–2955, 2023.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. In *Advances in Neural Information Processing Systems*, 2020.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021.

Lujie Yang, Hongkai Dai, Alexandre Amice, and Russ Tedrake. Approximate optimal controller synthesis for cart-poles and quadrotors via sums-of-squares. *IEEE Robotics and Automation Letters*, 2023.

Lujie Yang, Hongkai Dai, Zhouxing Shi, Cho-Jui Hsieh, Russ Tedrake, and Huan Zhang. Lyapunov-stable neural control for state and output feedback: A novel formulation. In *Forty-first International Conference on Machine Learning*, 2024.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, pp. 4944–4953, 2018.

Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane S. Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020.

Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *arXiv preprint arXiv:2208.05740*, 2022.

Hengjun Zhao, Xia Zeng, Taolue Chen, Zhiming Liu, and Jim Woodcock. Learning safe neural network controllers with barrier certificates. *Formal Aspects of Computing*, 33:437–455, 2021.

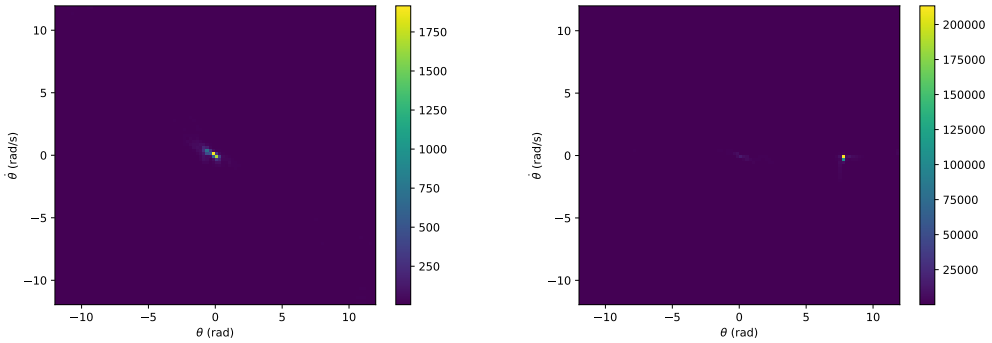
A Details of the Implementation and Experiments

We directly adopt the model architecture of all the controllers and Lyapunov functions from Yang et al. (2024) (we follow their source code which has some minor difference with the information provided in their paper). The controller is always a fully-connected NN with 8 hidden neurons in each hidden layer. For inverted pendulum and path tracking, there are 4 layers, and for 2D quadrotor, there are 2 layers. ReLU is used as the activation function. A NN-based Lyapunov function is used for inverted pendulum and path tracking, where the NN is a fully-connected NN with 4 layers, and the number of hidden neurons is 16, 16, and 8 for the three hidden layers, respectively. Leaky ReLU is used as the activation function for NN-based Lyapunov functions. A quadratic Lyapunov function with $n_r = 6$ is used for 2D quadrotor. For κ in Eq. (3), $\kappa = 0.001$ is used for inverted pendulum and path tracking, and $\kappa = 0$ is used for 2D quadrotor, following Yang et al. (2024).

We use a batch size of 30000 for all the training. We mainly use a learning rate of 5×10^{-3}, except 2×10^{-2} for path tracking. In the loss function, we set λ to 10^{-4}, λ_p to 0.1, and ϵ to 0.01. We try to make ρ_{ratio} as large as possible for individual systems, as long as the training works. We set $\rho_{\text{ratio}} = 0.1$ for 2D quadrotor. For inverted pendulum and path tracking, the range of ρ_{ratio} is between 0.5 and 0.9 for different settings. We start our dynamic splits after 100 initial training steps and continue until 5000 training steps (for 2D quadrotor) or if the training finishes before that (for other systems). For the adversarial attack, we use PGD with 10 steps and a step size of 0.25 relative to the size of subregion. We fix $\rho = 1.0$ during the training. At test time, we slightly reduce ρ to 0.9 for 2D quadrotor while we keep $\rho = 1.0$ for other systems. Using a slightly smaller ρ at test time instead of the value used for training has been similarly done in Yang et al. (2024) to ease the verification. Each training is done using a single NVIDIA GeForce RTX 2080 Ti GPU, while the verification with α,β-CROWN at test time is done on a NVIDIA RTX A6000 GPU which is the same GPU model used by Yang et al. (2024).

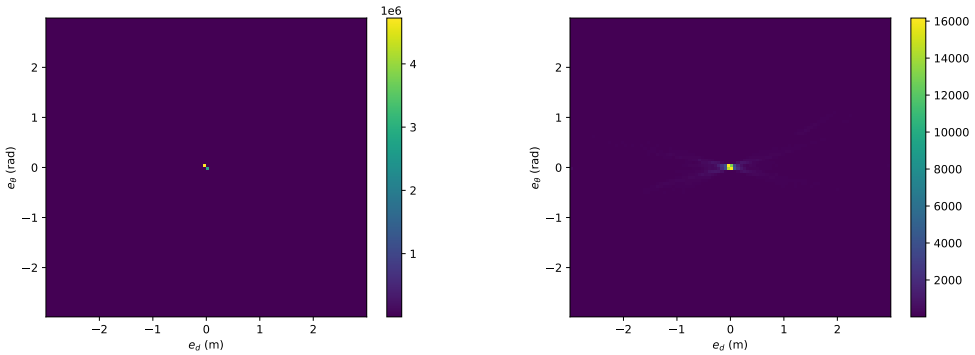
B Visualization of Branch-and-Bound

In this section, we visualize the distribution of subregions in the training dataset $\mathbb{D}$ at the end of the training, in order to understand where the most extensive branch-and-bound happens. Specifically, we check the distribution of the center of subregions. For systems with two input states (inverted pendulum and path tracking), we use 2D histogram plots, as shown in Figure 2 and 3. For the 2D quadrotor system which has 6 input states (and thus a 2D histogram plot cannot be directly used), we plot the distribution for different measurements of the subregion centers, including the ℓ_1 norm, ℓ_∞ norm, and the minimum magnitude over all the dimensions, as shown in Figure 4. We find that much more extensive splits tend to happen when at least one of the input states is close to that of the equilibrium state. Such areas have relatively small Lyapunov function values and tend to be more challenging for the training and verification. Specifically, in Figure 2a, 3a and 3b, extensive splits happen right close to the equilibrium state, while in Figure 2b, although extensive splits are not fully near the equilibrium state, extensive splits happen for subregions where the value for the $\dot{\theta}$ input state is close to 0 (i.e., value of $\dot{\theta}$ for the equilibrium state). The observation is also similar for the 2D quadrotor system, where Figure 4c shows that most subregions have at least one input state close to 0.



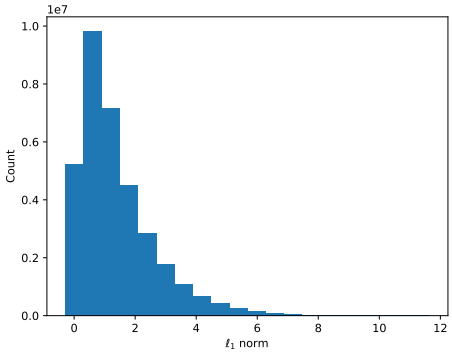
(a) Large torque. (b) Small torque.

Figure 2: Visualization for the distribution of subregions in $\mathbb{D}$ at the end of the training for the inverted pendulum system, with large torque limit and small torque limit, respectively. The 2D histogram plots show the distribution of the center of subregions. θ and $\dot{\theta}$ denote the angular position and angular velocity, respectively, for the two input states in inverted pendulum.

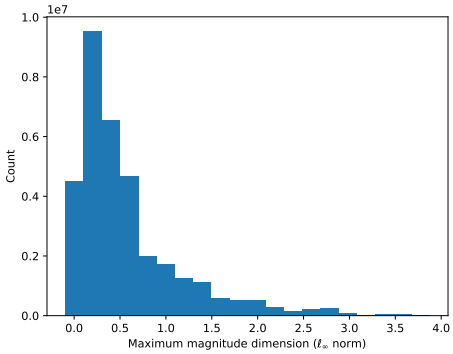


(a) Large torque. (b) Small torque.

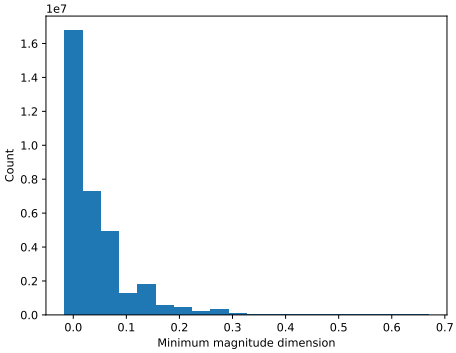
Figure 3: Visualization for the distribution of subregions in $\mathbb{D}$ at the end of the training for the path tracking system, similar to Figure 2. e_d and e_θ denote the distance error and angle error, respectively, for the two input states in path tracking.



(a) ℓ_1 norm.



(b) ℓ_∞ norm.



(c) Minimum magnitude over all the dimensions.

Figure 4: Visualization for the distribution of subregions in $\mathbb{D}$ at the end of the training for the 2D quadrotor system. We check the distribution of ℓ_1 norm, ℓ_∞ norm, and the minimum magnitude over all the dimensions (all the input states), respectively, for the subregion centers.