Combining Neural Networks and Symbolic Regression for Analytical Lyapunov Function Discovery

Jie Feng [*1] **Haohan Zou** [*1] **Yuanyuan Shi** [1]

Abstract

We propose CoNSAL (Combining Neural networks and Symbolic regression for Analytical Lyapunov function) to construct analytical Lyapunov functions for nonlinear dynamic systems. This framework contains a neural Lyapunov function and a symbolic regression component, where symbolic regression is applied to distill the neural network to precise analytical forms. Our approach utilizes symbolic regression not only as a tool for translation but also as a means to uncover counterexamples. This procedure terminates when no counterexamples are found in the analytical formulation. Compared with previous results, CoNSAL directly produces an analytical form of the Lyapunov function with improved interpretability in both the learning process and the final results. We apply CoNSAL to 2-D inverted pendulum, path following, Van Der Pol Oscillator, 3-D trig dynamics, 4-D rotating wheel pendulum, 6-D 3-bus power system, and demonstrate that our algorithm successfully finds their valid Lyapunov functions. Code examples are available at github.

1. Introduction

The field of deep learning has sparked significant interest in learning and data-driven control techniques for nonlinear systems. Yet, a major hurdle for the practical implementation of learning-based methods is their lack of guaranteed stability or safety, alongside interoperability issues (Amodei et al., 2016). To address this challenge, a common solution involves identifying valid certification functions for nonlinear dynamical systems. A well-known certificate function for stability guarantees is the Lyapunov function (Khalil, 2002), which is an energy-based function used to prove the stability of the equilibrium point. The Lyapunov functions are indispensable for control system designers, enabling to affirm the stability of complex dynamics and offer vital insights into system behaviors. The significance of such functions has led to the development of various computational construction methods (Giesl & Hafstein, 2015), where (McGough et al., 2010) deploys an evolutionary algorithm for symbolic computation of Lyapunov functions and (Ahmadi & Majumdar, 2016; Dai & Permenter, 2023) use sum-of-squares (SOS) methods. However, these methods either require pre-defined function templates or lack flexibility due to limited candidates.

Recent advancements in learning-based methods have paved the way for successful data-driven techniques in the discovery of neural-network-based Lyapunov functions, as demonstrated in studies such as (Edwards et al., 2024; Wang et al., 2024; Yang et al., 2024; Wu et al., 2023; Zhou et al., 2022; Chang et al., 2019). We refer readers to a recent review (Dawson et al., 2023) for more information. Despite the advancements, current approaches face two major challenges: 1) Generalization and 2) Scalable Verification (Dawson et al., 2023). The issue of generalization arises from the fact that these learned certificates are trained on specific dynamics within a specific region on fixed parameters. As a result, it is not trivial to generalize the learned Lyapunov functions to a greater state space or adapt to slightly different dynamics. Furthermore, an effective solution for verifying certificates in networks remains elusive, as the existing formal verification methods including satisfiability modulo theories (SMT) (Chang et al., 2019), mixed-integer linear programming (MIP) (Wu et al., 2023), and α, β-CROWN (Yang et al., 2024) can be computationally expensive, which restricts their applicability in complex systems.

On the contrary, analytical Lyapunov functions have two benefits: 1) It is interpretable and can potentially guarantee global stability, which enhances its generalizability and may provide further insights to scientists. Specifically, a stability-guaranteed control policy can be designed with a known analytical Lyapunov function (Feng et al., 2023a;b; Cui et al., 2023b). 2) It can bypass the neural network verification. Building on the preceding discussion, our research focuses on the following question: *Can method based on*

[*]Equal contribution [1]Department of Electrical and Computer Engineer, UC San Diego, La Jolla, United States. Correspondence to: Jie Feng <jif005@ucsd.edu>.

Workshop on Foundations of Reinforcement Learning and Control at the 41st International Conference on Machine Learning, Vienna, Austria. Copyright 2024 by the author(s).

Neural Networks directly yields analytical Lyapunov functions for nonlinear dynamical systems?

To tackle this challenge, we propose CoNSAL, short for Combining Neural networks and Symbolic regression (Angelis et al., 2023) for Analytical Lyapunov function, which comprises a neural network learner and a falsifier similar to (Chang et al., 2019). Unlike this method, which utilizes SMT solvers for validation, we employ symbolic regression to elucidate the neural Lyapunov function in analytical form and verify stability conditions with its roots. Stability is ascertained if the nonnegative Lyapunov function's only root in a specified region corresponds to the desired equilibrium point (such as the origin), and its nonpositive Lie derivative either has no roots or a single root at its equilibrium. Global stability can be concluded when the conditions are globally valid. If the stability condition is not met, we efficiently generate counterexamples near the root, adding them to the training set to update the neural Lyapunov function. The procedure terminates once a valid Lyapunov function is found. Given that numerical root-finding algorithms do not provide guarantees, a formal verification can be applied to the final analytical Lyapunov function we found, which is a lot easier compared with verifying neural networks. Following the compositional neural certificate (Zhang et al., 2023), we further propose a compositional neural Lyapunov function design to scale up our approach to networked systems.

We evaluate our algorithm with various nonlinear systems including the pendulum system, path-following problem, Van Der Pol oscillator, rotating wheel pendulum, 3-bus power system, and problems from nonlinear systems textbook (Khalil, 2002). Our method demonstrated the efficiency and robustness in discovering valid analytical Lyapunov functions. Contributions can be summarized as follows:

- We present the first algorithm that combines neural networks with symbolic regression to construct analytical Lyapunov functions directly, which significantly facilitates interpretability.
- We propose an efficient counterexample generation paradigm by sampling around the roots of the symbolic Lyapunov functions, which circumvents expensive neural network verifiers.
- We validate our algorithm across various examples with neural networks with more than 100 neurons.

2. Related Works

2.1. Learning-based Lyapunov function construction

The area of learning-based Lyapunov function construction is experiencing significant growth. A prominent model in this field is the Neural Lyapunov Control (Chang et al., 2019). This framework includes a neural-network-based Lyapunov function and a learnable linear controller, with stability verified by an SMT solver. This methodology was extended by (Zhou et al., 2022), which introduced a neural network to model unknown dynamics and another for control. For switched affine systems, (Feng et al., 2024) introduces a joint learning scheme for a common Lyapunov function and the controller. Additionally, (Dai et al., 2021; Wu et al., 2023) consider discrete-time systems and neural network controllers, verifying stability via MIP solvers. One limitation of these methods is the need for linearization of dynamics or other relaxations for formal verification. (Yang et al., 2024) deploys α, β-CROWN for neural network verification due to its scalability, and extends the previous control scheme from state feedback to output feedback control. However, these formal verification methods face challenges with large neural networks: SMT solvers can handle up to 30 neurons, MIP solvers are limited to networks with up to 200 neurons (Dawson et al., 2023), and (Yang et al., 2024) has 16 neurons each layer for the Lyapunov neural network.

2.2. Symbolic regression

Symbolic regression is a supervised machine-learning technique that constructs analytical models to represent datasets. It typically addresses a multi-objective optimization problem, balancing between minimizing prediction error and model complexity. Common practice involves employing genetic algorithms for solution finding, a brute-force approach whose computational complexity scales up exponentially with input dimension (Schmidt & Lipson, 2010). To address this concern, (Cranmer et al., 2020) proposes to combine neural networks with the genetic algorithm-based symbolic regression tool *eureqa* (Schmidt & Lipson, 2009). This method first learns a black-box model and then applies symbolic regression to find the mathematical formulation for scientific discovery. An alternative to generic algorithm-based techniques is "SINDy" (Sparse Identification of Nonlinear Dynamics) (Brunton et al., 2016), which formulates expressions as linear combinations from a fixed dictionary of nonlinear functions. Additionally, (Petersen et al., 2021; Shojaee et al., 2023) introduces deep generative models for symbolic regression and can achieve fast inference.

Given the extensive array of available algorithms, as summarized in (Cranmer, 2023), we selected the open-source *PySR* (Cranmer, 2023) for symbolic regression in our algorithm. This choice is driven by *PySR*'s user-friendly nature, its ability to optimize unknown constants, the provision of multiple solutions beyond mere accuracy, and the flexibility to customize operators and handle constraints. Other packages in the community like "SINDy" could also likely be used and achieve similar results.

3. Preliminary

3.1. Lyapunov Stability

We consider the problem of constructing Lyapunov functions for autonomous nonlinear dynamical systems at an equilibrium point. Without loss of generality, we assume the equilibrium point is the origin.

Definition 3.1 (Dynamical systems)**.** An n-dimensional autonomous nonlinear dynamical system is defined as

$$\frac{dx}{dt} = f(x)\,, x(0) = x_0, \tag{1}$$

where $f : \mathcal{D} \to \mathbb{R}^n$ is a Lipschitz-continuous vector field, and $\mathcal{D} \subseteq \mathbb{R}^n$ is an open set containing the origin that defines the state space. Each $x(t) \in \mathcal{D}$ is a state vector.

Definition 3.2 (Asymptotic stability)**.** The origin of system (1) is stabilized if for any $\epsilon > 0$, there exists $\delta(\epsilon) > 0$ such that $\|x(t)\| < \epsilon, \forall t \geq 0$ if $\|x(0)\| < \delta$. The origin is asymptotically stable if it is stable and δ can be chosen such that $\|x(0)\| < \delta \implies \lim_{t\to\infty} x(t) = 0$ (Khalil, 2002).

Definition 3.3 (Lie derivative)**.** The Lie derivative of a continuously differentiable scalar function $V : \mathcal{D} \to \mathbb{R}$ along the trajectory of (1) is

$$L_f V(x) = \sum_{i=1}^{n} \frac{\partial V}{\partial x_i}\frac{dx_i}{dt} = \sum_{i=1}^{n} \frac{\partial V}{\partial x_i} f_i(x). \tag{2}$$

Proposition 3.4 (Lyapunov functions for asymptotic stability)**.** *Let* 0 *be an equilibrium point for* (1) *and* $\mathcal{D} \subseteq \mathbb{R}^n$ *be a domain containing the origin. Let* $V : \mathcal{D} \to \mathbb{R}$ *be a continuously differentiable function such that*

$$V(0) = 0 \text{ and } V(x) > 0 \text{ in } D\backslash\{0\}, \tag{3a}$$

$$L_f V(x) \leq 0 \text{ in } D. \tag{3b}$$

Then, $x = 0$ *is stable. Moreover, if*

$$L_f V(x) < 0 \text{ in } D\backslash\{0\}, \tag{4}$$

then the origin is asymptotically stable.

3.2. Neural Lyapunov Network

Following (Kolter & Manek, 2019), a neural network parameterization for Lyapunov function is provided as follows,

$$V_\phi(x) = \sigma(g(F(x)) - g(F(0))) + \epsilon\|x\|_2^2, \tag{5}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a continuously differentiable invertible function, $g(x)$ is an input-convex neural network (ICNN) (Amos et al., 2017), ϵ is a small positive constant, $\sigma(\cdot)$ is a smoothed ReLU function defined by

$$\sigma(x) = \begin{cases} 0 & \text{if } x \leq 0, \\ \frac{x^2}{2d} & \text{if } 0 < x < d, \\ x - \frac{d}{2} & \text{otherwise} \end{cases}$$

Following Proposition 3.4, the Lyapunov function needs to be positive semi-definite and have no local optima except the origin (due to the negative Lie derivative condition (4)). Convexity of the neural network can prevent the existence of local optima. To enlarge the search space to potential non-convex candidates, function F is deployed as an input to ICNN $g(\cdot)$. The final Lyapunov function is shifted to guarantee positive semi-definiteness.

The neural Lyapunov function can be updated by minimizing the Lyapunov risk (Chang et al., 2019) defined as

$$L(\phi) = \frac{1}{N}\sum_{i=1}^{N} \left(max(0, L_f V_\phi(x_i))\right), \tag{6}$$

where $x_i \in \mathcal{D}\,, \forall i \in [1, 2, 3, ..., N]$ are counterexamples that violate the stability condition (4). By design of the neural Lyapunov function, $max(0, -V_\phi(x)) = V_\phi^2(0) = 0$, thus we only need to enforce the Lie derivative condition.

3.3. Symbolic Regression

We use the symbolic regression package *PySR* for deriving neural network's analytical expressions, which employs a genetic algorithm to stochastically assemble algebraic operators to fit a mathematical model to the given dataset.

With a predefined set of operators, *PySR* proposes a closed-form analytical expression at each complexity level, maximizing the probability of finding the most appropriate expression. For example, complexity 1 might yield x^2, complexity 2 results in $2x^2$, and complexity 3 produces $2x^2 + y$, continuing up to the maximum depth. To align with the characteristics of the Lyapunov function, the operators used in our study include $+, -, *, /$, square, $\sin, \cos$, and $1 - \cos$, where $1 - \cos$ is deployed as the non-negative counterparts of $\cos$. Our approach allows for the nesting of these operators and the optimization of constants. Expressions given by *PySR* may look like $\sin(\sin(x - y))$ or $0.64 * (\sin(x))^2 * (1 - \cos(y)) + 0.57z$, where 0.64 and 0.57 are examples of the constants optimized by *PySR*.

4. Proposed Method

In this section, we detail our proposed framework which can directly construct an analytical Lyapunov function, and thus can be rigorously verified locally and potentially generalized globally. We summarize the training paradigm in Algorithm 1 and present the framework in Figure 1.

4.1. Framework Overview

We start by introducing the outlook for our framework. As demonstrated in Figure 1, our framework contains two loops. The inner loop is the learner, which optimizes the neural network defined by (5) through minimizing the Lyapunov risk

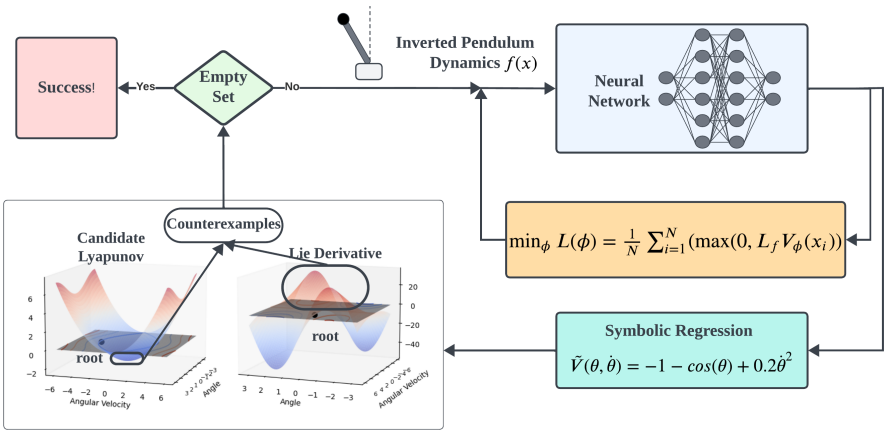

Figure 1: Diagram illustrating the proposed framework applied to an inverted pendulum. The analytical form (invalid intermediate result) enables root identification for both the candidate Lyapunov function and the Lie derivative. Counterexamples are sought around the roots. The process concludes when no counterexamples are detectable.

(6). The outer loop is the falsifier, which deploys symbolic regression to retrieve an analytical formulation for the neural network, and then generates counterexamples for further optimization. The outer loop is triggered every N_{in} updates of the inner loop to avoid redundant regressions. This process, including the N_{in} updates and the symbolic regression, is referred to as one epoch. For networked systems with high-dimensional states, we adopt the compositional neural Lyapunov function design in (Zhang et al., 2023) for scalability. In the remaining part, we detail the symbolic-regression-based falsifier and the compositional design.

4.2. Symbolic Regression and Falsification

To facilitate the training of the neural Lyapunov network, we use the symbolic regression package *PySR* to derive the neural network's analytical expressions. The process begins by uniformly sampling N_p points within the state space, which are input into the neural network to generate corresponding outputs. These input-output pairs are then fed into *PySR*, which provides n symbolic formulations at different complexity levels modeling the neural network. Notably, those N_p pairs are solely for fitting the symbolic regression model, not for neural network training.

With the expressions proposed by *PySR*, we evaluate the candidate Lyapunov functions with the Symbolic Python (SymPy) (Meurer et al., 2017) for direct computation of the closed-loop form derivative. Specifically, given $\tilde{V}(x)$ provided by *PySR*, we directly calculate $\frac{d\tilde{V}(x)}{dx}$. This allows us to get the Lie derivative following (2) with the known dynamics $f(x)$. Subsequently, we employ root-finding tools to identify roots for both the Lyapunov function and the Lie derivative. For root identification of both $\tilde{V}(x)$ and $L_f\tilde{V}(x)$, we apply SciPy's *fsolve* function (Virtanen et al., 2020) in its default setting. If nonzero roots are detected, the Lyapunov function is invalidated. Otherwise, we sample a random nonzero state $x \in \mathcal{D}$ to verify that $\tilde{V}(x) > 0$, which rules out the condition that the whole Lyapunov function $\tilde{V}(x)$ is negative without roots in the considered region. A similar operation can validate $L_f\tilde{V}(x) < 0$. With a successful pass of the two simple verifications, $\tilde{V}(x)$ is a numerically valid Lyapunov function for dynamics $f(x)$ in the interested domain. Notably, although our neural network design inherently ensures positive semi-definiteness, we do not restrict PySR to only produce PSD formulations. Non-PSD formulations can be valuable, sometimes requiring just constant optimization to get the valid Lyapunov function.

When nonzero roots are located, counterexamples can be generated from these roots. For instance, if a nonzero root r satisfies $L_f\tilde{V}(r) = 0$, it is typically feasible to find $L_f\tilde{V}(r+\epsilon) > 0$ with some small enough ϵ. Counterexamples are then created by progressively increasing ϵ until $L_f\tilde{V}(r+\epsilon) < 0$. While both (3a) and (4) are checked for the candidate Lyapunov functions, we only gather counterexamples violating (4) for network learning. Additionally, we randomly sample points in the state space to identify other potential counterexamples. Recall that PySR proposes multiple different candidate functions, we generate counterexamples for each and combine them for the subsequent neural network training. The algorithm terminates once a numerically valid analytical Lyapunov function is found. We present a pseudocode of the proposed learning framework for Lyapunov function construction in Algorithm 1.

Remark 4.1. Without a perfect root finder, our method accelerates counterexample generation, leading to an analytical Lyapunov function. In experiments, we apply SMT to the found Lyapunov functions for final validation.

Algorithm 1 CoNSAL (Combining Neural networks and Symbolic regression for Analytical Lyapunov function)

Input: Dynamics $f(x)$, Training steps N_{in}, Maximum complexity n.
Initialize the neural Lyapunov function $V_\phi(x)$,
Randomly sample input points $x \in \mathcal{D}$,
repeat
  for $i = 1$ **to** N_{in} **do**
    $L_f V_\phi = \sum_{i=1}^{n} \frac{\partial V_\phi}{\partial x_i} f_i(x)$, {Forward Pass}
    Compute Lyapunov risk $L(\phi)$,
    $\phi \leftarrow \phi - \eta \nabla_\phi L(\phi)$, {Back propagation}
  end for
  $\{\tilde{V}_1, \cdots \tilde{V}_n\} \leftarrow$ PySR(V_ϕ), {Symbolic Regression}
  for $j = 1$ **to** n **do**
    Find roots for $\tilde{V}_j$, {Falsification}
    if Lyapunov conditions (3a) and (4) hold, **then**
      Lyapunov function $\tilde{V}_j$ is valid, return.
    end if
    Generate counterexamples,
  end for
  Concatenate x with counterexamples generated by $\tilde{V}_j$ for all $j \in [1, 2, 3...n]$,
until Lyapunov function is valid.

4.3. Compositional Neural Lyapunov Function

Although this proposed algorithm can efficiently find the Lyapunov functions for relatively small systems, it's still an open challenge for neural Lyapunov functions to scale up to high-dimensional systems, e.g. large-scale network systems. In this paper, we consider networked systems involving m subsystems $\mathcal{S} = [1, 2, ..., m]$ following (Zhang et al., 2023), and the dynamics of each subsystems can be written as $\frac{dx_i}{dt} = f_i(x_i, x_{i_1}, x_{i_2}, ..., u_i) = f_i(x_i, x_{\mathcal{N}_i}, u_i)$, where $\mathcal{N}_i$ means the neighbor of subsystem i including $i_1, i_2, ...$ that are connected to subsystem i. With slight violation of notations, we use $x_i \in \mathbb{R}^{d_i}$ as the states of subsystem i and u_i as the corresponding actions.

To address this issue, an intuitive way is to find compositional Lyapunov functions for the high-dimensional system instead of a single Lyapunov function, i.e. finding one Lyapunov function $V_i(x_i)$ for each subsystem i. Moreover, if the networked system has a similar symmetric structure for different subsystems, the local Lyapunov functions can share the same structure across different subsystems. Following (Zhang et al., 2023), when the individual Lyapunov functions $V_i(x_i)$ satisfy an Input-to-State Stability (ISS)-style condition, they will certify the stability of the entire dynamical system.

Inspired by these results, we propose generalizing the compositional neural Lyapunov function design to certify stability. This approach ensures stability by considering the interrelationships between subsystems in addition to the individual Lyapunov functions for each subsystem. Instead of using one shared neural network to learn $V_i(x_i)$, we propose to include a supplementary neural network that models the interactions between different subsystems, i.e. $V_{ij}(x_i, x_j)$. $V_{ij}(x_i, x_j)$ can be regarded as the energy of the edge that connects subsystems i, j. For simplicity, we assume that each edge only connects two subsystems, each subsystem has the same state dimensions and shares symmetric structures, and the Lyapunov function for the whole system can be represented by $V(x) = \sum_{i \in \mathcal{S}} c_i V_i(x_i) + \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{N}_i} c_{ij} V_{ij}(x_i, x_j)$, where c_i, c_{ij} are some constants. These assumptions are satisfied by the power system dynamics (Cui et al., 2023b) and the Platoon system (Zhang et al., 2023). In this case, we achieve a significant reduction of input dimensions, which simplifies the training of neural networks and the following symbolic regression. Specifically, we model $V_\theta(x_i)$ with parameter θ for all $V_i(x_i)$ and $V_\psi(x_i, x_j)$ with parameter ψ for $V_{ij}(x_i, x_j)$, c_i, c_{ij} are also set as learnable parameters. To reduce the number of neural networks, parameters of neural network that models $V_i(\cdot)$ are shared across m subsystems and so do those modeling $V_{ij}(\cdot, \cdot)$. The neural network structure is presented in Figure 2

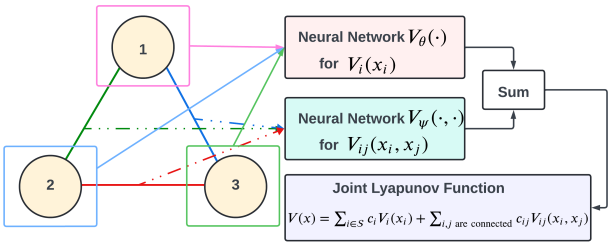

Figure 2: Diagram illustrating the proposed compositional structure for a three-node network. Each subsystem's states are fed to the neural network $V_\theta(\cdot)$, and the connected subsystems' states are fed to $V_\psi(\cdot, \cdot)$. These neural networks are shared across edges and subsystems, and the joint Lyapunov function for the entire system is obtained by summation.

We train and validate the compositional Lyapunov function with a similar scheme as the low-dimensional systems. During training, we optimize the Lyapunov risk (6) with the joint Lyapunov function $V(x)$ to jointly update the neural Lyapunov functions and the coefficients c_i, c_{ij}. After several updating steps, symbolic regression is triggered to reveal the analytical formulation of $V_i(x_i)$ and $V_{ij}(x_i, x_j)$, which are then combined with the coefficients to get $V(x)$. The joint Lyapunov function is then validated by the same process. If it is not valid, we generate counterexamples around the roots and continue the iterative training.

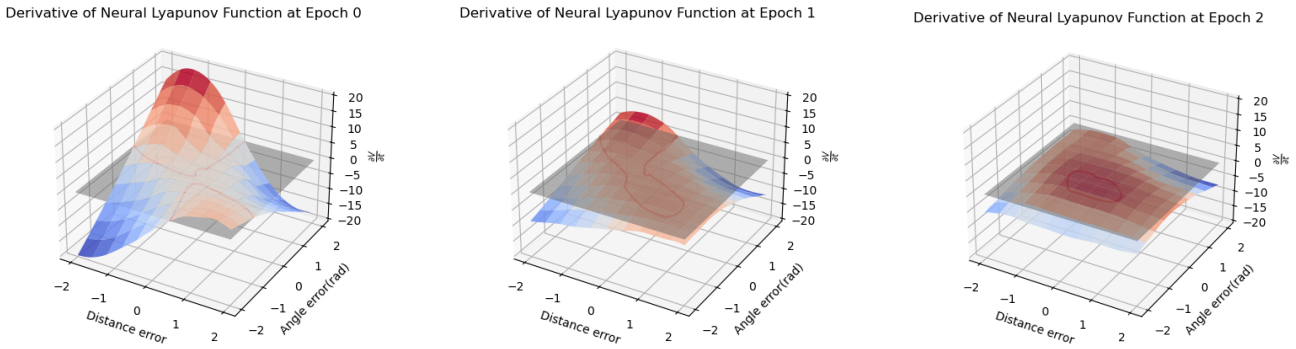

Figure 3: Lie Derivative of the Neural Lyapunov Function under linear path following dynamics, updated with the proposed algorithm. Showing the result of the first two epochs.

5. Experiment

In this section, we validate the proposed algorithm with various nonlinear dynamics by finding their Lyapunov functions, where the autonomous systems (without additional control) are stable at the origin. We test the proposed algorithm using the neural Lyapunov network (5) on 2-D path following, inverted pendulum, Van Der Pol oscillator, 3-D trig dynamics, 4-D rotating wheel pendulum, and 6-D nonlinear dynamics. The compositional neural Lyapunov function is applied to a 3-bus power system. To ease the requirement for symbolic regression to find the exact constants in the Lyapunov functions and accommodate the error of numerical root-finding tools, we allow a violation of the stability conditions less than 10^{-4} in value and the Lyapunov function is only verified at a given local region numerically. CoNSAL aims to find an analytically correct Lyapunov function with tolerable errors. Global stability can be assessed through further inspection. In this work, multiple initial guesses are applied to the root finder to facilitate root finding.

We first demonstrate the evolution of the Neural Lyapunov function, and compare how the proposed framework optimizes the neural Lyapunov function with the SMT-based scheme (Chang et al., 2019) in the 2D path following environment, where our approach can achieve better performance with respect to the SMT solver. The runtime information and the found analytical Lyapunov functions are summarized in Table 2. Details about dynamics are available in Appendix B. Neural network configurations and base operator selection are available in Appendix C.

5.1. Evolution of Neural Lyapunov Function

Linear path following is a classic nonlinear control task, which controls a wheeled robot to follow a linear path with dynamics (7) given by (Samson, 1992):

$$\dot{x}_1 = v \cdot sin(x_2), \tag{7a}$$

$$\dot{x}_2 = -x_2 - c \cdot v \cdot \frac{sin(x_2)}{x_2} \cdot x_1, \tag{7b}$$

where x_1, x_2 are the distance and the and angel difference between the robot and the reference line, $v \in \mathbb{R}^+$ is a constant velocity and $c \in \mathbb{R}^+$ is a positive real constant. Specifically, we set $c = 2, v = 6\,\mathrm{m\,s}^{-1}$. We consider state space $\mathcal{D} = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \leq 2, |x_2| \leq \pi\}$. This system is stable to the origin and has a known Lyapunov function $V(x_1, x_2) = x_1^2 + \frac{x_2^2}{c}$, which can guarantee global asymptotic stability. With this system, we illustrate the evolution of the neural Lyapunov function in Figure 3. As shown in the figure, the landscape of the Lie derivative evolves rapidly to satisfy the stability condition, and the violation of the negative Lie derivative decreases efficiently, demonstrating the effectiveness of our proposed framework.

We further compare our framework with the SMT-based neural Lyapunov function (Chang et al., 2019), where the SMT solver is deployed to generate counterexamples for the neural network and validate the stability conditions. We use the same two-layer neural networks with 128 hidden units for both methods. Given the relatively large neural network, the SMT-based approach can only achieve 30% successful rate for converging, while ours achieved 80 % successful rate for finding the valid symbolic Lyapunov function. Following Table 1, due to the complexity of the neural network, the SMT solver consumes much more time compared with symbolic-regression-based verification.

The counterexamples generated by both algorithms are presented in Figure 4. Following this result, the SMT solver can identify a small region such that samples from the region will violate the stability constraint. Thus counterexamples are generated by randomly sampling in this region. However, this method provides counterexamples focusing on this small region and can potentially lead to overfitting. With

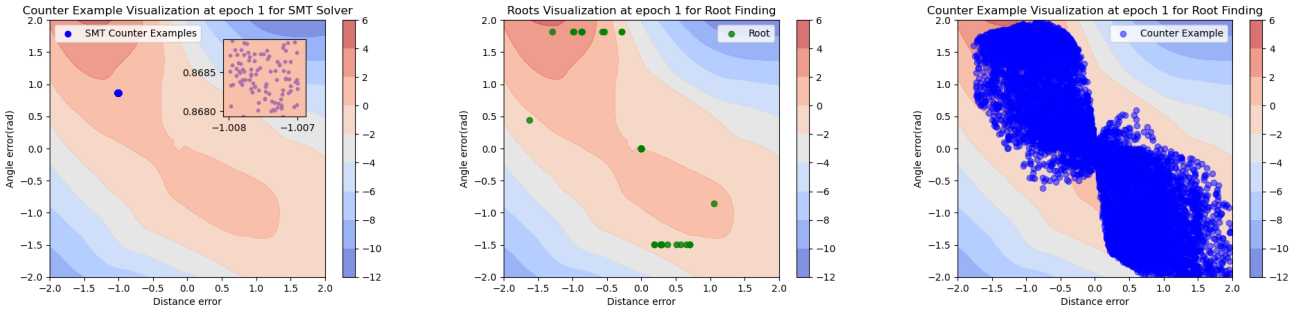

Figure 4: SMT Counter Examples, Roots, and Root finding Counter Examples Visualization using the same checkpoint at epoch 1 for Lie derivative of the neural Lyapunov functions under linear path-following dynamics. The same number of counterexamples are generated. The zoomed-in region shows the counterexamples from SMT solvers.

PySR, multiple symbolic expressions are generated at different complexity levels. The roots of these formulations can be efficiently found numerically, which are illustrated in the middle plot of Figure 4. With acceptable approximating errors, the roots outline the region of violation and the counterexamples can be efficiently generated near the roots. In this case, our approach finds counterexamples covering most of the violation areas in the state space and facilitates the successful discovery of a valid Lyapunov function.

Table 1: Runtime and Performance Comparison

Verification Methods Comparison		
Metric \ **Method**	Ours	SMT Solver
Verification Time	5.81s	92.08s
Convergence Time	1401s	5051s
Convergence Epoch	45	58
Success Rate	80%	30%

Note: Verification time measures the average time consumption for each single-step verification. Only success trials' convergence time and convergence epoch are included in this table. The success rate is calculated out of 10 different trials. An experiment that found a valid Lyapunov function within 100 epochs is considered a successful trial.

We conclude this comparison with the loss curve in Figure 5. Given that our approach covers a larger violation space than the SMT-based method, our algorithm can induce a greater loss calculated by (6) and a larger variance. However, our method can still achieve comparable loss convergence compared with the SMT-based approach. Empirically we also find that, if the SMT-based neural Lyapunov function fails to converge, PySR cannot find the analytical Lyapunov function from the SMT-trained neural network even if it has a low Lyapunov risk.

5.2. Runtime and Found Analytical Lyapunov Functions

We summarize the runtime and the found Lyapunov functions in table 2. In general, the proposed framework can achieve more than 60% success rate for all the tested dynamics and find at least a local Lyapunov function within 50 symbolic regression calls, where a local Lyapunov function means the Lyapunov function is valid in the tested local

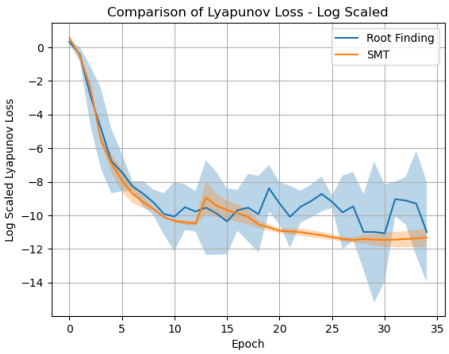

Figure 5: Log-scaled Loss Convergence Comparison using loss (6). Only success trials are considered.

region. Global stability is further extended by human experts. Notably, our algorithm demonstrates fast convergence for the tested 4-D and 6-D systems. We also find that our method can occasionally discover new Lyapunov functions valid locally (detailed in B.2). Here, 'new' means that the Lyapunov function is not included in the existing literature.

However, it gets the lowest success rate with the 2-D inverted pendulum system and the 6-D 3-bus power system. The challenge comes from optimizing the constants in the Lyapunov function based on given physical constants in the dynamics. Considering the inverted pendulum system, its local Lyapunov function can be represented as $V(x_1, x_2) = (1 - cos(x_1)) + \frac{l}{2g}x_2^2$, where x_1 is the angular position from the inverted position, x_2 is the angular velocity, and g, m, l, b are acceleration of gravity, mass of inverted object, the length of string, and the coefficient of friction respectively. Setting $l = 5\text{m}$, $g = 9.81m/s^2$, it is challenging for the symbolic regression to find the correct coefficients. In failed cases, our framework often finds the correct structure but incorrect coefficients, e.g. $V = 1 - cos(x_1) + 0.26x_2^2$, which is not valid.

In cases where prior dynamics knowledge is available, we incorporate this knowledge into the Lyapunov function design

Dynamics	Time	Epoch	Found Lyapunov Functions[a]	Stability[‡]	Succ % [‡]
2-D Path Following (Linear)	1401s	45	$V = x_1^2 + \frac{x_2^2}{c}$	g.a.s.	80%
2-D Inverted Pendulum	1356s	37	$V = 1 - cos(x_1) + 0.254x_2^2$	l.a.s.	60%
2-D Van Der Pol Oscillator	230s	7	$V = x_1^2 + x_2^2$	l.a.s.	100%
3-D Trig Dynamics	1216s	41	$V = 2sin(x_1)^2 + x_2^2 + 2sin(x_3)^2$	l.a.s.	100%
4-D Rotating Wheel Pendulum	1556s	34	$V = E^2 + x_3^2 + x_4^2$ *	g.a.s.	80%
6-D Nonlinear Dynamics	916s	13	$V = 4x_1^2 - 1.24x_2^2 + \left(\sum_{i=1}^{5} x_i\right)^2 + 2.996\left(0.578\sum_{i=2}^{6} x_i^2 + 1\right)^2$	l.a.s.	100%
6-D 3-bus Power System	271s	3	$V = 0.516\left(\sum_{i=1}^{3} \omega_i^2\right) - 0.5\sum_{i=1}^{3}\sum_{\substack{j=1 \\ i\neq j}}^{3} (cos(\delta_i - \delta_j) - 1)$	l.a.s.	60%

a. All Found Lyapunov functions passed SMT solver's verification for Lyapunov conditions, with tolerable error $\epsilon = 10^{-4}$.

$*$. E is the energy function of the corresponding dynamics, which is defined at B.4.

‡. In this column, 'g.a.s' represents globally asymptotically stable, and 'l.a.s.' represents locally asymptotically stable.

‡. 'Succ %' denotes the successful rate of finding a valid Lyapunov function out of 5 random seeds.

Only success trials' convergence time and convergence epoch are included in this table.

Table 2: Training time and the found analytical Lyapunov functions for the proposed algorithm with different dynamics.

to ease this task. Specifically, for the 4-D rotating wheel pendulum, we use the energy function E as part of the input to the neural Lyapunov function, along with all state variables. As a result, a Lyapunov based on the energy function is revealed by our method. Details of all tested dynamics and considered local regions are provided in Appendix B.

6. Conclusion

In this paper, we propose CoNSAL, a framework to construct analytical Lyapunov functions for stable nonlinear dynamics. Our approach integrates a neural network learner with a symbolic regression solver, which extracts an analytical formula from the neural Lyapunov function. This symbolic Lyapunov function is further validated with numerical root-finding approaches. When the Lyapunov function is invalid, we efficiently sample counterexamples around the roots to facilitate the training process. The resulting valid Lyapunov function can potentially demonstrate global stability or aid experts in designing controllers. Compared to existing work, CoNSAL provides an analytical Lyapunov function instead of neural ones, which significantly improves the interpretability and generalizability of the final results. Moreover, it bypasses the need for neural network verification, enabling the use of larger neural networks and allowing for scalability to more complex systems. The efficiency of the proposed framework is validated in multiple nonlinear dynamical systems by successfully finding a valid local Lyapunov function for these test cases.

Our results open up several exciting future directions: (1) Efficiently incorporating known constants from the dynamics into symbolic regression to reveal analytical formulations without scaling up the input size; (2) Exploring neural network-based symbolic regression methods for fast inference and end-to-end training in Lyapunov function discovery without the requirement for base operators; (3) Investigating formal verification for analytical formulations beyond numerical root-finding; (4) Scaling up to larger dynamical systems with more than 10 dimensions; (5) Integrating a nonlinear controller in our framework and optimizing it alongside the Lyapunov function in an end-to-end manner, ensuring stability guarantees; (6) Maximizing the region of attraction. These directions can enhance the efficiency and generalizability of CoNSAL, making it applicable to the vast cases where analytical Lyapunov functions are valuable for validation, analysis, or controller design.

Acknowledgements

This work was supported under grants NSF ECCS-2200692 and Jacobs School Early Career Faculty Development Award 2023. Jie Feng is supported by the UC-National Laboratory In-Residence Graduate Fellowship L24GF7923. Haohan Zou is funded by UC San Diego Halıcıoğlu Data Science Institute (HDSI) scholarship. Yuanyuan Shi is partially supported by the Hellman Fellowship.

References

Ahmadi, A. A. and Majumdar, A. Some applications of polynomial optimization in operations research and real-time decision making. *Optimization Letters*, pp. 709–729, 2016.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete problems in ai safety, 2016.

Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *International Conference on Machine Learning*, pp. 146–155. PMLR, 2017.

Angelis, D., Sofos, F., and Karakasidis, T. E. Artificial intelligence in physical sciences: Symbolic regression trends and perspectives. *Archives of Computational Methods in Engineering*, pp. 1–21, 2023.

Brunton, S. L., Proctor, J. L., and Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

Chang, Y.-C., Roohi, N., and Gao, S. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

Cranmer, M. Interpretable machine learning for science with pysr and symbolicregression.jl, 2023.

Cranmer, M., Sanchez Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., and Ho, S. Discovering symbolic models from deep learning with inductive biases. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17429–17442. Curran Associates, Inc., 2020.

Cui, W., Jiang, Y., and Zhang, B. Reinforcement learning for optimal primary frequency control: A lyapunov approach. *IEEE Transactions on Power Systems*, 38(2):1676–1688, 2023a. doi: 10.1109/TPWRS.2022.3176525.

Cui, W., Jiang, Y., Zhang, B., and Shi, Y. Structured neural-PI control with end-to-end stability and output tracking guarantees. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b.

Dai, H. and Permenter, F. Convex synthesis and verification of control-lyapunov and barrier functions with input constraints. In *2023 American Control Conference (ACC)*, pp. 4116–4123. IEEE, 2023.

Dai, H., Landry, B., Yang, L., Pavone, M., and Tedrake, R. Lyapunov-stable neural-network control. In *Proceedings of Robotics: Science and Systems*, 2021.

Dawson, C., Gao, S., and Fan, C. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics*, 39(3):1749–1767, 2023. doi: 10.1109/TRO.2022.3232542.

Edwards, A., Peruffo, A., and Abate, A. Fossil 2.0: Formal certificate synthesis for the verification and control of dynamical models, 2024.

Fantoni, I. and Lozano, R. The reaction wheel pendulum. 01 2002. doi: 10.1007/978-1-4471-0177-2_7.

Feng, J., Cui, W., Cortés, J., and Shi, Y. Bridging transient and steady-state performance in voltage control: A reinforcement learning approach with safe gradient flow. *IEEE Control Systems Letters*, 7:2845–2850, 2023a. doi: 10.1109/LCSYS.2023.3289435.

Feng, J., Shi, Y., Qu, G., Low, S. H., Anandkumar, A., and Wierman, A. Stability constrained reinforcement learning for decentralized real-time voltage control. *IEEE Transactions on Control of Network Systems*, pp. 1–12, 2023b. doi: 10.1109/TCNS.2023.3338240.

Feng, J., Muralidharan, M., Henriquez-Auba, R., Hidalgo-Gonzalez, P., and Shi, Y. Stability-constrained learning for frequency regulation in power grids with variable inertia. *IEEE Control Systems Letters*, 8:994–999, 2024.

Giesl, P. and Hafstein, S. Review on computational methods for lyapunov functions. *Discrete and Continuous Dynamical Systems - B*, 20(8):2291–2331, 2015. ISSN 1531-3492. doi: 10.3934/dcdsb.2015.20.2291.

Grüne, L. Computing lyapunov functions using deep neural networks. *Journal of Computational Dynamics*, 8, 01 2019. doi: 10.3934/jcd.2021006.

Khalil, H. K. *Nonlinear systems*. Prentice Hall, 3nd edition, 2002.

Kolter, J. Z. and Manek, G. Learning stable deep dynamics models. *Advances in neural information processing systems*, 32, 2019.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

McGough, J. S., Christianson, A. W., and Hoover, R. C. Symbolic computation of lyapunov functions using evolutionary algorithms. In *Proceedings of the 12th IASTED international conference*, volume 15, pp. 508–515, 2010.

Meurer, A., Smith, C. P., Paprocki, M., Čertík, O., Kirpichev, S. B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J. K.,

Singh, S., Rathnayake, T., Vig, S., Granger, B. E., Muller, R. P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M. J., Terrel, A. R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., and Scopatz, A. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103.

Petersen, B. K., Larma, M. L., Mundhenk, T. N., Santiago, C. P., Kim, S. K., and Kim, J. T. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.

Samson, C. Path following and time-varying feedback stabilization of a wheeled mobile robot. *Second International Conference on Automation, Robotics and Computer Vision*, 3, 01 1992.

Schmidt, M. and Lipson, H. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. doi: 10.1126/science.1165893.

Schmidt, M. and Lipson, H. *Symbolic Regression of Implicit Equations*, pp. 73–85. Springer US, Boston, MA, 2010. ISBN 978-1-4419-1626-6. doi: 10.1007/978-1-4419-1626-6_5.

Shojaee, P., Meidani, K., Barati Farimani, A., and Reddy, C. Transformer-based planning for symbolic regression. In *Advances in Neural Information Processing Systems*, volume 36, pp. 45907–45919. Curran Associates, Inc., 2023.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wang, Z., Andersson, S. B., and Tron, R. Lyapunov neural network with region of attraction search, 2024.

Wu, J., Clark, A., Kantaros, Y., and Vorobeychik, Y. Neural lyapunov control for discrete-time systems. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Yang, L., Dai, H., Shi, Z., Hsieh, C.-J., Tedrake, R., and Zhang, H. Lyapunov-stable neural control for state and output feedback: A novel formulation for efficient synthesis and verification, 2024.

Zhang, S., Xiu, Y., Qu, G., and Fan, C. Compositional neural certificates for networked dynamical systems. In Matni, N., Morari, M., and Pappas, G. J. (eds.), *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, volume 211 of *Proceedings of Machine Learning Research*, pp. 272–285. PMLR, 15–16 Jun 2023.

Zhou, R., Quartz, T., Sterck, H. D., and Liu, J. Neural lyapunov control of unknown nonlinear systems with stability guarantees. In *Advances in Neural Information Processing Systems*, 2022.

A. Code Availability

The code for our proposed algorithm CoNSAL is available at https://github.com/HaohanZou/CoNSAL.

B. Dynamics of the Tested Systems

B.1. Inverted Pendulum

The inverted pendulum is a well-known classical nonlinear system that contains two state variables. The dynamics are formulated as follows,

$$\begin{aligned}\dot{x}_1 &= x_2,\\ \dot{x}_2 &= -\frac{g}{l}sin(x_1) - \frac{b}{m}x_2,\end{aligned}$$

where x_1 is the angular position from the inverted position, x_2 is the angular velocity, and parameters g, m, l, b are acceleration of gravity, the mass of the inverted object, the length of string, and the coefficient of friction respectively. In experiment, setting $g = 9.81$, $m = 2\,\mathrm{kg}$, $l = 5\,\mathrm{m}$, and $b = 0.1$, our proposed method finds the valid Lyapunov function $V = 1 - cos(x_1) + 0.254x_2^2$ over the state space: $\mathcal{D} = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \leq \pi \text{ and } |x_2| \leq 6\}$. This found Lyapunov function has the same analytical structure as the energy function of the inverted pendulum.

B.2. Van Der Pol Oscillator

Van Der Pol Oscillator is a nonconservative, oscillating system with nonlinear damping (Zhou et al., 2022). The dynamics of the Van Der Pol Oscillator have two state variables and are formulated as follows,

$$\begin{aligned}\dot{x}_1 &= x_2,\\ \dot{x}_2 &= -x_1 - \mu(1 - x_1^2) \cdot x_2,\end{aligned}$$

where x_1 and x_2 represent the object's position in the Cartesian coordinate, parameter $\mu \in \mathbb{R}^+$ indicates the strength of the damping. Under the state space $\mathcal{D} = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_i| \leq 1\}$ and setting $\mu = 1$, our proposed method found valid local Lyapunov function $V(x_1, x_2) = x_1^2 + x_2^2$. Other forms of Lyapunov functions for Van Der Pol Oscillator, for example, $V(x_1, x_2) = (x_1 \cdot (x_1 + x_2) + x_2^2)/(2.8 - x_1 - x_2)$, are also recovered during the experiments.

B.3. 3-D Trig Dynamics

3-D trig dynamics comes from exercise problems from textbook (Khalil, 2002), the dynamics are written as follows,

$$\begin{aligned}\dot{x}_1 &= x_2,\\ \dot{x}_2 &= -2h(x_1) - x_2 - 2h(x_3),\\ \dot{x}_3 &= x_2 - x_3,\end{aligned}$$

where $h(x) = sin(x)cos(x)$. When the state space is $\mathcal{D} = \{(x_1, x_2, x_3) \in \mathbb{R}^3 \mid |x_i| \leq 1.5, \forall i \in \{1, 2, 3\}\}$, the valid Lyapunov function found by our proposed method is $V(x_1, x_2, x_3) = 2sin(x_1) + x_2^2 + 2sin(x_3)^2$, which is consistent to the textbook solution of Lyapunov function for this particular dynamics.

B.4. Rotating Wheel Pendulum

The rotating wheel pendulum is structured as a pendulum with an additional rotating wheel at the end string, which freely spins along an axis parallel to the axis of the pendulum (Fantoni & Lozano, 2002). The dynamics of the system have four state variables and are formulated as follows,

$$
\begin{aligned}
\dot{x}_1 &= x_2, \\
\dot{x}_2 &= \frac{d_{22}}{det(D)}\bar{m}g \cdot x_1 + \frac{-d_{12}}{det(D)} \cdot \tau, \\
\dot{x}_3 &= x_4, \\
\dot{x}_4 &= \frac{d_{21}}{det(D)}\bar{m}g \cdot x_1 + \frac{d_{11}}{det(D)} \cdot \tau, \\
\tau &= \frac{-x_4 - x_3 + k_1 sin(x_1)}{E + k_2},
\end{aligned}
$$

where x_1 is the angular position from the inverted position of the pendulum, x_2 is the angular velocity of the pendulum, x_3 is the angular position of the wheel, x_4 is the angular velocity of the wheel, τ is the torque applied to disk from the motor, m_1 is the mass of the pendulum, m_2 is the mass of the wheel, l_1 is the length of the pendulum, l_2 is the distance to the center of mass of the pendulum, I_1 is the moment of inertia of the pendulum, I_2 is the moment of inertia of the wheel, $\bar{m} = m_1 l_2 + m_2 l_1$, $k_1 = \frac{d_{21}\bar{m}g}{det(D)}$, and $k_2 = \frac{d_{11}}{det(D)}$. $E = \frac{1}{2}\begin{bmatrix} x_2 & x_4 \end{bmatrix} D \begin{bmatrix} x_2 \\ x_4 \end{bmatrix} + \bar{m}g(cos(x_1) - 1)$ is the energy function of the dynamics, where D is the inertia matrix formulated as

$$
D = \begin{bmatrix} m_1 l_2^2 + m_2 l_1^2 + I_1 + I_2 & I_2 \\ I_2 & I_2 \end{bmatrix}.
$$

We consider the state space as $\mathcal{D} = \{(x_1, x_2, x_3, x_4) \in \mathbb{R}^4 \mid |x_1| \leq \frac{\pi}{2}, |x_3| \leq \frac{\pi}{2}, |x_2| \leq 2, \text{ and } |x_4| \leq 2\}$.

With control law $\tau = \frac{-x_4 - x_3 + k_1 sin(x_1)}{E + k_2}$, the system is stable. For this system, directly learning a Lyapunov function is difficult. Thus, we investigate the benefit of using prior knowledge of the system for constructing the Lyapunov function. Specifically, we use the energy function E as the additional information on the dynamics and use it as an additional variable to the neural Lyapunov function during training.

We set $m_1 = (\frac{0.1}{9.81})\text{kg}$, $m_2 = (\frac{0.4}{9.81})\text{kg}$, $l_1 = l_2 = 1\,\text{m}$, $I_1 = (1 - \frac{0.5}{9.81})\text{kg} \cdot \text{m}^2$, and $I_2 = 1\text{kg} \cdot \text{m}^2$, the proposed method found a valid Lyapunov function $V(x_1, x_2, x_3, x_4, E) = E^2 + x_3^2 + x_4^2$.

B.5. 6-D Nonlinear Dynamics

This high-dimensional dynamics consists of three two-dimensional asymptotically stable linear subsystems that are coupled by three nonlinearities with small gains adopted from (Grüne, 2019). The dynamics are written as

$$
\begin{aligned}
\dot{x}_1 &= -x_1 + 0.5x_2 - 0.1x_5^2, \\
\dot{x}_2 &= -0.5x_1 - x_2, \\
\dot{x}_3 &= -x_3 + 0.5x_4 - 0.1x_1^2, \\
\dot{x}_4 &= -0.5x_3 - x_4, \\
\dot{x}_5 &= -x_5 + 0.5x_6, \\
\dot{x}_6 &= -0.5x_5 - x_6 + 0.1x_2^2.
\end{aligned}
$$

Our proposed method is able to find a valid Lyapunov function $V(x) = 4x_1^2 - 1.24x_2^4 + (x_1 + x_2 + x_3 + x_4 + x_5)^2 + 2.996 \cdot (0.578x_2^2 + 0.578x_3^2 + 0.578x_5^2 + 0.578x_6^2 + 1)^2$ over the region $\mathcal{D} = \{(x_1, x_2, x_3, x_4, x_5, x_6) \in \mathbb{R}^6 \mid |x_i| \leq 1, \forall i \in \{1, 2, .., 6\}\}$, which is a valid Lyapunov function for the given dynamics.

B.6. Power System

We test the compositional neural Lyapunov function design with a 3-bus power system (Cui et al., 2023a) to examine its ability on high-dimensional dynamics. Consider θ_i, ω_i as the phase angle and the frequency of bus i, respectively, the

dynamics for each bus are formulated as follows,

$$\dot{\theta}_i = \omega_i,$$
$$m_i\dot{\omega} = p_i - d_i\omega_i - u_i(\omega_i) - \sum_{j=1}^{3} B_{ij} sin(\theta_i - \theta_j),$$

where m_i is the generator inertia constant, d_i is the combined frequency response coefficient from synchronous generators and frequency sensitive load, and p_i is the net power injection, for each bus $i = 1, 2, 3$. $B \in \mathbb{R}^{3\times 3}$ is the susceptance matrix with $B_{ij} = 0$ for every pair $\{i, j\}$ such that bus i and bus j are not connected, and $u_i(\omega_i)$ is the controller at bus i that adjusts the power injection to stabilize the frequency.

Since the frequency dynamics of the system depends only on the phase angle differences, so we change the coordinates:

$$\delta_i = \theta_i - \frac{1}{3}\sum_{i=1}^{3}\theta_i$$

where δ_i can be understood as the center-of-inertia coordinates of each bus. In our experiment, for simplicity, we set $p_i = 0$, $m_i = 1$, $d_i = 1$, $u_i(\omega_i) = \omega_i$, and $B_{ij} = 1 \; \forall \; i \neq j, B_{ii} = 0$. In this case, the equilibrium point for our system is at the origin, i.e., $\delta_i^* = \omega_i^* = 0$, $i = 1, 2, 3$. The state space for our experiment is defined as: $\mathcal{D} = \{(\delta_1, \delta_2, \delta_3, \omega_1, \omega_2, \omega_3) \in \mathbb{R}^6 \mid |\delta_i| \leq \frac{\pi}{4}$ and $|\omega_i| \leq 2$ for $i = 1, 2, 3\}$.

For the training procedures, we composed and jointly trained two neural networks for these dynamics. First, for each distinct bus i, $(\delta_i, \omega_i) \in \mathbb{R}^2$ are fed into an Input Convex Neural Network (ICNN). Then, for each pair of connected buses $\{i, j\}$, we input their frequency differences $\delta_i - \delta_j$ into another feed-forward Neural Network. In the end, symbolic regression is applied to each distinct neural network, and found Lyapunov function is the composition of regression formulas from two neural networks.

Through our method, we retrieved a valid Lyapunov function $V(\delta_1, \delta_2, \delta_3, \omega_1, \omega_2, \omega_3) = 0.516\left(\sum_{i=1}^{3}\omega_i^2\right) - 0.5\sum_{i=1}^{3}\sum_{\substack{j=1\\ i\neq j}}^{3}(cos(\delta_i - \delta_j) - 1)$ from the joint training of two neural networks.

C. Neural Network Structure and Symbolic Regression Setting

We now detail the neural network structures and the base operators used for symbolic regression in the following table. For simplicity, we set $F(\cdot)$ defined in (5) as $F(x) = x$.

Dynamics	**Neural Lyapounov Function**	**Symbolic Regression Unary Operator**
2-D Path Following (Linear)	(2, 128, 128, 1)	$[sin, cos, 1 - cos, x^2]$
2-D Inverted Pendulum	(2, 128, 128, 1)	$[sin, cos, 1 - cos, x^2]$
2-D Van Der Pol Oscillator	(2, 128, 128, 1)	$[sin, cos, 1 - cos, x^2]$
3-D Trig Dynamics	(3, 128, 128, 1)	$[sin, cos, 1 - cos, x^2]$
4-D Rotating Wheel Pendulum	(4, 128, 128, 1)	$[sin, cos, 1 - cos, x^2]$
6-D Nonlinear Dynamics	(6, 128, 128, 1)	$[x^2]$
6-D 3-bus Power System	(2, 128, 128, 1) & (1, 32, 32, 1)	$[x^2]$ & $[x^2, cos, 1 - cos]$

Table 3: Network size and regression model set up for each task. The tuple denotes the number of neurons in each layer of Neural Network. The set denotes the pre-defined operators for Symbolic Regression model. For all experiments, we set binary operators as $[+, -, \times, \div]$.