LyZNet: A Lightweight Python Tool for Learning and Verifying Neural Lyapunov Functions and Regions of Attraction

Jun Liu
j.liu@uwaterloo.ca
University of Waterloo
Waterloo, Canada

Yiming Meng
ymmeng@illinois.edu
University of Illinois
Urbana, United States

Maxwell Fitzsimmons
mfitzsimmons@uwaterloo.ca
University of Waterloo
Waterloo, Canada

Ruikun Zhou
ruikun.zhou@uwaterloo.ca
University of Waterloo
Waterloo, Canada

ABSTRACT

In this paper, we describe a lightweight Python framework that provides integrated learning and verification of neural Lyapunov functions for stability analysis. The proposed tool, named *LyZNet*, learns neural **Ly**apunov functions using physics-informed neural **net**works (PINNs) to solve **Z**ubov's equation and verifies them using satisfiability modulo theories (SMT) solvers. What distinguishes this tool from others in the literature is its ability to provide verified regions of attraction close to the domain of attraction. This is achieved by encoding Zubov's partial differential equation (PDE) into the PINN approach. By embracing the non-convex nature of the underlying optimization problems, we demonstrate that in cases where convex optimization, such as semidefinite programming, fails to capture the domain of attraction, our neural network framework proves more successful. The tool also offers automatic decomposition of coupled nonlinear systems into a network of low-dimensional subsystems for compositional verification. We illustrate the tool's usage and effectiveness with several numerical examples, including both non-trivial low-dimensional nonlinear systems and high-dimensional systems.

CCS CONCEPTS

• **Mathematics of computing** → **Ordinary differential equations**; • **Computing methodologies** → **Neural networks**; • **Theory of computation** → *Automated reasoning*.

KEYWORDS

Nonlinear systems, stability analysis, neural networks, formal verification, satisfiability modulo theories, interval analysis

ACM Reference Format:
Jun Liu, Yiming Meng, Maxwell Fitzsimmons, and Ruikun Zhou. 2024. LyZNet: A Lightweight Python Tool for Learning and Verifying Neural Lyapunov Functions and Regions of Attraction. In *27th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '24), May 14–16, 2024, Hong Kong SAR, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3641513.3650134

HSCC '24, May 14–16, 2024, Hong Kong SAR, China

ACM ISBN 979-8-4007-0522-9/24/05
https://doi.org/10.1145/3641513.3650134

1 INTRODUCTION

Stability analysis of nonlinear dynamical systems has been a focal point of research in control and dynamical systems. In many applications, characterizing the domain of attraction for an asymptotically stable equilibrium point is crucial. For example, in power systems, understanding the domain of attraction is essential for assessing whether the system can recover to a stable equilibrium after experiencing a fault.

Since Lyapunov's landmark paper more than a hundred years ago [31], Lyapunov functions have become a cornerstone of nonlinear stability analysis, providing an instrumental tool for performing such analyses and estimating the domain of attraction. Consequently, extensive research has been conducted on Lyapunov functions. One of the key technical challenges is the construction of Lyapunov functions. To address this challenge, both analytical [18, 42] and computational methods [14, 15] have been investigated.

Among computational methods for Lyapunov functions, sums-of-squares (SOS) techniques have garnered widespread attention [21, 36–38, 43, 44]. These methods facilitate not only local stability analysis but also provide estimates of regions of attraction [36, 43, 44]. Leveraging semidefinite programming (SDP), one can extend the region of attraction by employing a specific "shape function" within the estimated region. However, selecting such shape functions in a principled manner, beyond standard norm [36] or quadratic functions [24], remains elusive.

On the other hand, Zubov's theorem [48] precisely characterizes the domain of attraction through a partial differential equation (PDE). This contrasts with commonly seen Lyapunov conditions, which manifest as partial differential inequalities. The use of an equation, rather than inequalities, enables precise characterization of the domain of attraction. The concept of maximal Lyapunov function [45] is closely related to Zubov's method. The authors of [45] have also provided a computational procedure for constructing maximal Lyapunov functions using rational functions.

1.1 Related work

The proposed tool solves Zubov's equation using physics-informed neural networks (PINNs) [25, 41] and verifies the neural solutions using satisfiability modulo theories (SMT) solvers. The tool currently relies on dReal [13] for verification through interval analysis.

Computation of Lyapunov functions using Zubov's equation has been previously explored, for example, with radial basis functions [14] and SOS techniques [21] (though the authors did not explicitly mention Zubov's equation). More recently, the authors of [22] employed a data-driven approach to solve Zubov's equation using neural networks; however, the computed neural solutions were not verified, and Zubov's equation was not directly encoded in the loss

function. The authors of [27] solved Zubov's equation using neural networks and verified them with dReal, but did not contrast their results with SOS techniques.

Many authors have recently investigated the use of neural networks for computing Lyapunov functions (see, e.g., [2, 4, 12, 12, 17, 22], and [9] for a recent survey). In fact, such efforts date back to as early as the 1990s [30, 40]. Unlike SDP-based synthesis of SOS Lyapunov functions, neural network Lyapunov functions obtained by training are not guaranteed to be Lyapunov functions. Subsequent verification is required, e.g., using satisfiability modulo theories (SMT) solvers [3, 4]. The use of SMT solvers for searching and refining Lyapunov functions has been explored previously [23]. Counterexample-guided search of Lyapunov functions using SMT solvers is investigated in [3] and the associated tool [1], which supports both Z3 [10] and dReal [13] as verifiers. Neural Lyapunov functions with SMT verification are explored in [47] for systems with unknown dynamics. SMT verification is often time-consuming, especially when seeking a maximal Lyapunov function [27] or dealing with high-dimensional systems. Recent work has also focused on learning neural Lyapunov functions and verifying them through optimization-based techniques, e.g., [5–8]. Such techniques usually employ (leaky) ReLU networks and use mixed integer linear/quadratic programming (MILP/MIQP) for verification.

In contrast with existing approaches, the proposed tool offers the ability to compute regions of attraction (ROA) close to the domain of attraction and provides support for compositional verification to handle high-dimensional systems. We demonstrate its usage and effectiveness on both low-dimensional systems with non-trivial dynamics and high-dimensional systems to showcase compositional verification. The tool achieves state-of-the-art results on ROA approximations for nonlinear systems and their verification for high-dimensional systems.

2 PROBLEM FORMULATION

Consider a nonlinear system described by

$$\dot{x} = f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable. Suppose that the origin is an asymptotically stable equilibrium for the system. We denote the unique solution to (1) from the initial condition $x(0) = x_0$ by $\phi(t, x_0)$ for $t \in J$, where J is the maximal interval of existence for ϕ.

The *domain of attraction* of the origin for (1) is defined as

$$\mathcal{D} := \left\{ x \in \mathbb{R}^n : \lim_{t \to \infty} \|\phi(t, x)\| = 0 \right\}. \tag{2}$$

We know that $\mathcal{D}$ is an open and connected set.

We call any forward invariant subset of $\mathcal{D}$ a *region of attraction* (ROA). We are interested in computing regions of attraction, as they not only provide a set of initial conditions with guaranteed convergence to the equilibrium point, but also ensure constraints and safety through forward invariance.

Lyapunov functions provide an indispensable tool for stability analysis and ROA estimates. A *Lyapunov function* for asymptotic stability of the origin is a continuously differentiable function satisfying $V(0) = 0$, $V(x) > 0$ for all $x \neq 0$, and the derivative of V along solutions of (1) (sometimes called its Lie derivative) satisfies

$$\dot{V}(x) := \frac{dV}{dx} \cdot f(x) < 0, \quad \forall x \neq 0.$$

Given any such function and any positive constant $c > 0$, the sublevel set $\{x \in \mathbb{R}^n : V(x) \leq c\}$ is a region of attraction, provided that the set is contained in $\mathcal{D}$.

The proposed tool is designed to accomplish the following task: given an asymptotically stable equilibrium point, it computes neural Lyapunov functions that can certify regions of attraction with provable guarantees.

What sets our tool and approach apart from existing ones in the literature is its ability to verify regions of attraction close to the domain of attraction by incorporating Zubov's equation into the training of neural Lyapunov functions.

3 TOOL OVERVIEW AND USAGE

3.1 Tool overview

The tool consists of a set of Python modules built around the machine learning framework PyTorch [39] and the numerical satisfiability modulo theories (SMT) solver dReal [13]. It also relies on the NumPy library [19] for numerical computation, SciPy library [46] for scientific computing tasks, and the SymPy library [35] for symbolic calculation.

We describe the main modules as follows:

- `dynamical_systems.py` defines the `DynamicalSystems` class and associated methods such as linearization, computation of quadratic Lyapunov function by solving Lyapunov's equation, and providing interfaces for numerical and symbolic function valuations related to the vector field.
- `local_verifier.py` provides functions for verifying local stability using quadratic Lyapunov functions, with interval analysis from the SMT solver dReal. The key component is using the mean value theorem to bound the remainder of the linear approximation. This enables *exact* verification of local asymptotic stability, despite dReal's numerical conservativeness. This sets it apart from existing methods that often ignore verification around a neighbourhood of the origin.
- `quadratic_verifier.py` performs reachability analysis using a quadratic Lyapunov function. The target is the local ROA obtained above. The set guaranteed to reach this target is formulated as a sublevel set of the quadratic Lyapunov function.
- `generate_data.py` generates data for training neural Lyapunov functions. It involves solving system (1), augmented with an additional state to compute value for the solution to Zubov's equation, which we accomplish using SciPy and "embarrassing parallelization" with the joblib library [20].
- `neural_learner.py` learns a neural Lyapunov function by solving Zubov's PDE using physics-informed neural networks (PINN) [25, 41]. We allow customization of neural network structure as well as selection of different loss function modes.
- `neural_verifier.py` verifies the learned neural Lyapunov function indeed satisfies the Lyapunov condition required for reach a target invariant set within the domain of attraction. The target can be set as the region of attraction already verified by local

analysis or quadratic Lyapunov analysis above. Our current implementation uses dReal as the verifier.

- `network_verifier.py` includes functions for compositional verification of local stability using quadratic and neural Lyapunov functions. The underlying theory involves vector Lyapunov functions, differential inequalities, and comparison techniques for nonlinear systems.
- `sos_verifier.py`, `plot.py`, and `utils.py` provide support for comparisons with other Lyapunov functions, such as sums-of-squares (SOS) Lyapunov functions, visualization of verified level sets, and other routines used in learning and verification.

3.2 Tool usage

Using the tool is straightforward. As shown in the code snippet below, we can define a dynamical system with SymPy variables and symbolic expressions for the vector field. We can also specify the domain for learning and training, as well as name the system for file management and result storage.

```
import sympy
import lyznet

# Define dynamics
mu = 1.0
x1, x2 = sympy.symbols('x1 x2')
f_vdp = [-x2, x1 - mu * (1 - x1**2) * x2]
domain_vdp = [[-2.5, 2.5], [-3.5, 3.5]]
sys_name = "Van_der_Pol"
vdp_system = lyznet.DynamicalSystem(f_vdp,
    domain_vdp, sys_name)
```

The code above defines the reversed Van der Pol equation with parameter $\mu = 1.0$ on the domain $X = [-2.5, 2.5] \times [-3.5, 3.5]$.

The following lines of code can be used to call relevant functions for verifying local stability using quadratic Lyapunov function, as well as learning and verifying neural Lyapunov functions.

```
# Call the local stability verifier
c1_P = lyznet.local_stability_verifier(vdp_system)
# Call the quadratic verifier
c2_P = lyznet.quadratic_reach_verifier(vdp_system,
     c1_P)

# Generate data
data = lyznet.generate_data(vdp_system, n_samples
    =3000)

# Call the neural lyapunov learner
net, model_path = lyznet.neural_learner(vdp_system
    , data=data, lr=0.001, layer=2, width=30,
    num_colloc_pts=300000, max_epoch=20, loss_mode
    ="Zubov")

# Call the neural lyapunov verifier
c1_V, c2_V = lyznet.neural_verifier(vdp_system,
    net, c2_P)
```

Most functions and parameters are self-explanatory. We remark that the argument `loss_mode` in `neural_learner` currently can be set to three different modes: `"Zubov"`, `"Data"`, and `"Lyapunov"`. The latter two take a purely data-driven approach for solving Zubov's equation or encode a Lyapunov inequality instead of Zubov's equation, respectively.

The code above computes and verifies four sublevel sets:

$$\begin{aligned} \mathcal{P}_1 &= \{x \in X : V_P(x) \le c1\_P\}, \\ \mathcal{P}_2 &= \{x \in X : V_P(x) \le c2\_P\}, \\ \mathcal{V}_1 &= \{x \in X : V_N(x) \le c1\_V\}, \\ \mathcal{V}_2 &= \{x \in X : V_N(x) \le c2\_V\}, \end{aligned} \tag{3}$$

satisfying $\mathcal{P}_1 \subseteq \mathcal{P}_2, \mathcal{V}_1 \subseteq \mathcal{P}_2$, and $\mathcal{V}_1 \subseteq \mathcal{V}_2$, where V_P is a quadratic Lyapunov function of the form $V_P(x) = x^T P x$ and $V_N(x)$ is a learned neural network Lyapunov function. The set $\mathcal{P}_1$ is a local region of attraction verified by linearization. The set $\mathcal{P}_2$ represents the set of initial conditions from which solutions of (1) are verified to reach $\mathcal{P}_1$ using a quadratic Lyapunov function. The sets $\mathcal{V}_1$ and $\mathcal{V}_2$ are verified using a neural Lyapunov function, where $\mathcal{V}_1$ represents a target invariant set contained in $\mathcal{P}_2$[1], and $\mathcal{V}_2$ is an invariant set from which solutions of (1) are guaranteed to reach $\mathcal{V}_1$. All sets are also verified to be contained in the interior of X to ensure set invariance. Put together, $\mathcal{V}_2$ is a verified ROA for (1).

4 METHODOLOGY

4.1 Local stability analysis using quadratic Lyapunov functions

Local stability analysis using quadratic Lyapunov functions is fairly standard. Suppose that $x = 0$ is an exponentially stable equilibrium point for (1); i.e., $f(0) = 0$ and the Jacobian matrix $A = Df(0) = \frac{df}{dx}\big|_{x=0}$ is a Hurwitz matrix, i.e., all eigenvalues of A have negative real parts. Rewrite (1) as

$$\dot{x} = Ax + g(x), \tag{4}$$

where $g(x) = f(x) - Ax$ satisfies $\lim_{x\to 0} \frac{\|g(x)\|}{\|x\|} = 0$. Given any symmetric positive definite real matrix $Q \in \mathbb{R}^{n\times n}$, there exists a unique solution P to the Lyapunov equation

$$PA + A^T P = -Q. \tag{5}$$

Let $V_P(x) = x^T P x$. Then

$$\begin{aligned} \dot{V}_P(x) &= x^T (PA + A^T P)x + 2x^T P g(x) \\ &= -x^T Q x + 2x^T P g(x) \\ &\le -\lambda_{\min}(Q)\,\|x\|^2 + 2x^T P g(x) \\ &= -\varepsilon\,\|x\|^2 + (2x^T P g(x) - r\,\|x\|^2), \end{aligned}$$

where we set $r = \lambda_{\min}(Q) - \varepsilon > 0$ for some sufficiently small $\varepsilon > 0$ and $\lambda_{\min}(Q)$ is the minimum eigenvalue of Q. The goal of `local_stability_verifier` is to determine the largest number $c1\_P$ such that

$$x \in \mathcal{P}_1 \implies 2x^T P g(x) \le r\,\|x\|^2, \tag{6}$$

where $\mathcal{P}_1$ is as defind in (3).

While (6) can be easily formulated as a satisfiability condition in dReal [13], due to conservative use of interval analysis in dReal to account for numerical errors, verification of inequalities such as $2x^T P g(x) \le r\,\|x\|^2$ will return a counterexample close to the origin.

[1]This could be set to $\mathcal{P}_1$ if one wishes to skip the call of `quadratic_reach_verifier` for any reason, or any known region of attraction verified by other means.

To overcome this issue, we look at a higher-order approximation of $Pg(x)$. By the mean value theorem, we have

$$Pg(x) = Pg(x) - Pg(0) = \int_0^1 P \cdot Dg(tx)dt \cdot x,$$

where Dg is the Jacobian of g given by $Dg = Df - A$, which implies

$$2x^T Pg(x) \leq 2 \sup_{0\leq t\leq 1} \|P \cdot Dg(tx)\| \|x\|^2.$$

As a result, to verify (6), we just need to verify

$$x \in \mathcal{P}_1 \Longrightarrow 2 \sup_{0\leq t\leq 1} \|P \cdot Dg(tx)\| \leq r. \tag{7}$$

If $\mathcal{P}_1$ is star-shaped with respect to the origin[2], then (7) is equivalent to

$$x \in \mathcal{P}_1 \Longrightarrow 2\|P \cdot Dg(x)\| \leq r. \tag{8}$$

Since $Dg(0) = 0$ and Dg is continuous, one can always choose $c1\_P > 0$ sufficiently small such that (8) can be verified. Furthermore, in rare situations, if (8) can be verified for $X = \mathbb{R}^n$, then global exponential stability of the origin is proved for (1).

Remark 1. *From (8), one can further use easily computable norms, such as the Frobenius norm, to over-approximate the matrix 2-norm $\|P \cdot Dg(tx)\|$. Furthermore, by the implication $x^T Px \leq c1\_P \Longrightarrow |x_i| \leq \frac{c1\_P}{\lambda_{\min}(P)}$ for all $i = 1, \cdots, n$, and the use of a compositionally verifiable upper bound of the matrix 2-norm, such as the Frobenius norm, one can verify (8) in a compositional fashion. This idea was implemented in* `compositional_local_stability_verifier`*, which can work significantly more efficiently for high-dimensional systems. Of course, this decomposition of an ellipsoid set to a hyperrectangular set is inherently conservative. Less conservative results for compositional verification can be achieved with the help of vector Lyapunov functions [28], as implemented in* `network_verifier`*.*

4.2 Learning neural Lyapunov functions via Zubuv's equation for ROA verification

4.2.1 Zubov's theorem and maximal Lyapunov function. Zubov's PDE [48] takes the form

$$\dot{W}(x) := \nabla W(x) \cdot f(x) = -\Psi(x)(1 - W(x)), \tag{9}$$

where W is a positive definite function to be solved and Ψ is also a positive definite function that can be chosen. For instance, Ψ can be chosen as $\Psi(x) = \alpha(1 + W(x))\|x\|^2$, where $\alpha > 0$ is a parameter. Zubov's theorem states that if there exists a function W satisfying the above PDE on a given domain D containing the origin with the following two additional conditions: (1) $0 < W(x) < 1$ on D except $W(0) = 0$; (2) $W(x) \to 1$ as $x \to \partial D$, then the domain $D = \mathcal{D}$, the domain of attraction for the origin. Suppose that the function W is also trivially extended to the domain of interest X containing D with $W(x) = 1$ for $x \in X \setminus D$. We have $\mathcal{D} = \{x \in X : W(x) < 1\}$. In other words, the domain of attraction is characterized by the sublevel-1 set of the solution to Zubov's PDE.

[2]A set S is star-shaped with respect to the point x_0 if for all $y \in S$ the line segment joining y and x_0 is a subset of S.

Zubov's theorem is closely related to the notion of maximal Lyapunov function discussed in [45]. A maximal Lyapunov function on a domain D containing the origin satisfies the property

$$\dot{V}(x) := \nabla V(x) \cdot f(x) = -\omega(x), \tag{10}$$

for all $x \in D$, where ω is a positive definite function. Additionally, $V(x) \to \infty$ as $x \to \partial D$.

A solution W to Zubov's equation can be related to a maximum Lyapunov V by any strictly increasingly function $\beta : [0, \infty) \to \mathbb{R}$ that satisfies $\beta(0) = 0$ and $\beta(s) \to 1$ as $s \to \infty$. Indeed, given a maximal Lyapunov function V, we can simply choose

$$W(x) = \beta(V(x)). \tag{11}$$

There are obvious choices of such a function β. For examples $\beta(s) = 1 - \exp(-\alpha s)$ for $s \geq 0$ or $\beta(s) = \tanh(\alpha s)$ for $s \geq 0$, where $\alpha > 0$ is a parameter. In our implementation, we choose $\beta(s) = \tanh(\alpha s)$. It can be easily verified that W defined this way satisfies (9) with

$$\Psi(x) = -\alpha(1 + W(x))\omega(x). \tag{12}$$

Under suitable assumptions [45], a solution to (10) can be constructed as

$$V(x) = \int_0^\infty \omega(\phi(t, x))dt. \tag{13}$$

For example, if the origin is an exponentially stable equilibrium point and ω is locally Lipschitz, then this construction is valid and gives a maximal Lyapunov function on $\mathcal{D}$. One easy choice of ω is $\omega(x) = \|x\|^2$.

4.2.2 Physics-informed neural solution to Zubov's PDE. Put together, (9), (12), and (13) allow us to learn a neural Lyapunov function via physics-informed neural networks [25, 41] for solving PDEs.

In a nutshell, a physics-informed neural solution to the PDE (9) is a neural network function that minimizes the residual for satisfying (9). Additional conditions on this neural network function can be enforced (or rather encouraged) through additional loss terms.

More specifically, let $W_N(x; \theta)$ be a neural approximation to a solution of (9). The training loss consists of

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{residual}}(\theta) + \mathcal{L}_{\text{boundary}}(\theta) + \mathcal{L}_{\text{data}}(\theta), \tag{14}$$

where $\mathcal{L}_{\text{residual}}$ is the residual error of the PDE given by

$$\frac{1}{N}\sum_{i=1}^{N}(\nabla_x W_N(x_i; \theta)f(x_i) + \Psi(x_i)(1 - W_N(x_i; \theta)))^2, \tag{15}$$

evaluated over a set of collocation points $\{x_i\}_{i=1}^N$ chosen over a domain X. For instance, we can choose Ψ as in (12). The loss $\mathcal{L}_{\text{boundary}}$ captures boundary conditions. There are different ways boundary conditions can be added. The simplest one is that $W(0) = 0$ and $W(x) = 1$ for $x \notin \mathcal{D}$. Although optional, we can also encourage the following inequality [16, 27] near the origin:

$$\beta(c_1\|x\|^2) \leq W(x) \leq \beta(c_2\|x\|^2), \tag{16}$$

where $\beta(s) = \tanh(\alpha s)$ (corresponding to the choice of β above). This inequality is always satisfiable for some positive c_1 and c_2

when the origin is exponentially stable. Finally, the term $\mathcal{L}_{\text{data}}(\theta)$ captures data loss, which we define as

$$\mathcal{L}_{\text{data}}(\theta) = \frac{1}{N_d}\sum_{i=1}^{N_d}(W_N(y_i;\theta) - \hat{W}(y_i))^2, \tag{17}$$

where $\{\hat{W}(y_i)\}_{i=1}^{N_d}$ is a set of data points, which can be obtained by forward integration of (1) to obtain W from (13) and (11) [22].

4.2.3 Verification of regions of attraction. We briefly outline how local stability analysis via linearization and reachability analysis via a Lyapunov function can be combined to provide a ROA estimate that is close to the domain of attraction.

Let V_P and W_N be two generic functions. Suppose that V_P and $c > 0$ gives a local region of attraction $\{x \in X : V_P(x) \le c\}$. We can verify the following inequalities using SMT solvers:

$$(c_1 \le W_N(x) \le c_2) \wedge (x \in X) \implies \dot{W}_N(x) \le -\varepsilon, \tag{18}$$

$$(W_N(x) \le c_1) \wedge (x \in X) \implies V_P(x) \le c, \tag{19}$$

where $\varepsilon > 0, c_2 > c_1 > 0$. The first inequality ensures that solutions of (1) starting in $\{x \in X : W_N \le c_2\}$ always reaches the target set $\{x \in X : W_N \le c_1\}$, as long as the solution does not leave X. The second inequality ensures that the target set $\{x \in X : W_N \le c_1\}$ is contained in a local region of attraction $\{x \in X : V_P(x) \le c\}$. By definition, it follows that $\{x \in X : W_N \le c_2\}$ provides an underestimate of the true domain of attraction $\mathcal{D}$. An easy condition to ensure that solutions cannot leave X before reaching the target is that $\{x \in X : W_N \le c_2\}$ does not intersect with the boundary of X, which can also be easily verified by an SMT solver.

Now specific to the tool, we can use the above technique to first compute a region of attraction via linearization as in Section 4.1 and obtain $c1\_P$. We can then use the reachability conditions above to obtain an enlarged region of attraction, first using a quadratic Lyapunov function to verify $c2\_P$, and then using a neural Lyapunov function to verify $c1\_V$ and $c2\_V$, as shown in (3). If the neural Lyapunov function V_N is obtained by solving Zubov's PDE (9), then the set $\mathcal{V}_2 := \{x \in X : V_N(x) \le c2\_V\}$ provides a region of attraction close to the true domain of attraction.

Remark 2. *While there are numerous studies on computing neural Lyapunov functions [1–4, 12, 16, 47], the tools currently available cannot produce verified region of attraction estimates that are close to the state-of-the-art SOS approaches. In particular, we tested the recent tool Fossil 2.0 [1, 11] on Example 5.1 in the following section and found that the region of attraction verified by Fossil is worse than that verified by a quadratic Lyapunov function (dashed red curve in Figure 1) using LyZNet. On the other hand, in the next section, we demonstrate LyZNet's capability in outperforming state-of-the-art SOS approaches by providing less conservative verified region-of-attraction estimates. This is because LyZNet aims to solve Zubov's equation, which precisely characterizes the exact domain of attraction.*

5 EXAMPLES

In this section, we demonstrate the usage of the tool with several numerical examples. All experiments are conducted on a 2020 MacBook Pro with a 2 GHz Quad-Core Intel Core i5 and without any GPU. The purpose of this section is not to conduct extensive experiments, but to demonstrate the usage and effectiveness of the proposed tool. The equations describing the examples can be found in the Appendix. We note that the main purpose here is to demonstrate the capability of the tool. Readers are referred to [26, 28] for more detailed discussions of the numerical results and, especially, compositional verification techniques involved [28]. The repository of the tool includes code for running these examples (https://git.uwaterloo.ca/hybrid-systems-lab/lyznet).

Table 1: Parameters and verification results for Van der Pol equation (Example 5.1)

μ	Layer	Width	c2_V	Volume (%)	SOS volume (%)
1.0	2	30	0.871	**94.78%**	94.10%
3.0	2	30	0.642	**85.21%**	70.93%

Example 5.1 (Van der Pol equation). Consider the reversed Van der Pol equation with different values of μ. For $\mu = 1.0$ and $X = [-2.5, 2.5]\times[-3.5, 3.5]$, similar to the code snippet shown in Section 3.2, we train two neural networks of different sizes (depth and width). Figure 1 depicts the largest verifiable sublevel set, along with the learned neural Lyapunov function. It can be seen that the domain of attraction is quite comparable and slightly better than that provided by a sums-of-squares (SOS) Lyapunov function with a polynomial degree of 6, obtained using a standard "interior expanding" algorithm [36].

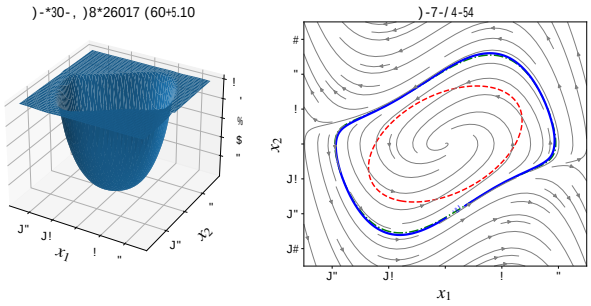

Figure 1: Verified neural Lyapunov function for Van der Pol equation with $\mu = 1.0$ (Example 5.1). Dashed red: quadratic Lyapunov function; dot-dashed green: SOS Lyapunov; solid blue: neural Lyapunov.

As the stiffness of the equation increases, we observe further improved advantages of neural Lyapunov approach over SOS Lyapunov functions. With $\mu = 3.0$ and domain $X = [-3.0, 3.0] \times [-6.0, 6.0]$, the comparison of neural Lyapunov function with SOS Lyapunov function is shown in Figire 2.

Example 5.2 (Two-machine power system). Consider a two-machine power system [45] which has an asymptotically stable equilibrium point at the origin and an unstable equilibrium point at $(\pi/3, 0)$. Figure 3 shows that a neural network with two hidden layers and 30 neurons in each layer provides a region-of-attraction estimate significantly better than that from a sixth-degree polynomial SOS

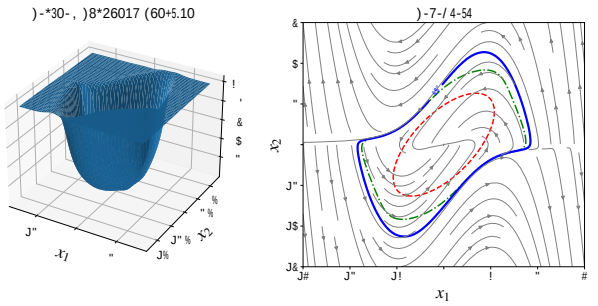

Figure 2: Verified neural Lyapunov function for Van der Pol equation with $\mu = 3.0$ (Example 5.1). Dashed red: quadratic Lyapunov function; dot-dashed green: SOS Lyapunov; solid blue: neural Lyapunov. The learned neural Lyapunov function outperforms a sixth degree SOS Lyapunov function.

Table 2: Parameters and verification results for a two-machine power system (Example 5.2)

Layer	Width	c2_V	Volume (%)	SOS volume (%)
2	30	0.740	**82.34%**	18.53%

Lyapunov function, computed with a Taylor expansion of the system model. The example shows that neural Lyapunov functions perform better than SOS Lyapunov functions when the nonlinearity is non-polynomial. We also compared with the rational Lyapunov function presented in [45], but the ROA estimate is worse than that from the SOS Lyapunov function, and we were not able to formally verify the sublevel set of the rational Lyapunov function reported in [45] with dReal [13]. Improving the degree of polynomial in the SOS approach does not seem to improve the result either.

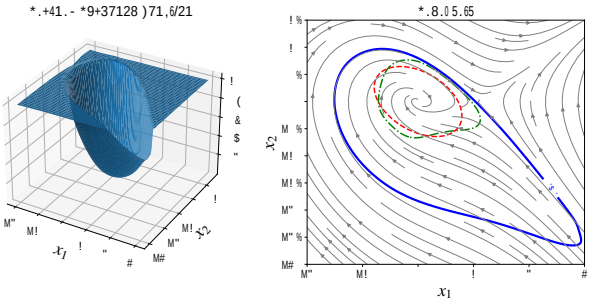

Figure 3: Verified neural Lyapunov function for a two-machine power system (Example 5.2). Dashed red: quadratic Lyapunov function; dot-dashed green: SOS Lyapunov; solid blue: neural Lyapunov. The learned neural Lyapunov function significantly outperforms a sixth degree SOS Lyapunov function.

Example 5.3 (Inverted pendulum with linear control). In the literature, there have been numerous references to this example as a benchmark for comparing techniques for stabilization with provable ROA estimates. Interestingly, our `local_stability_verifier` verifies global stability of the origin within 1 millisecond using dReal [13].

Remark 3. *Computational time for training and verification largely depends on computer hardware. On the somewhat outdated laptop used for experiments, training the neural network as shown in Tables 1 and 2 takes about 500 seconds. This is with 20 epochs and 300,000 collocation points using a minibatch size of 32. Verification, including all bisection subroutines, can range from 200 to 2,000 seconds. We report the computational times in the Appendix.*

Example 5.4 (10-dimensional system). The example was used by the authors of [16] and [12] to illustrate the training of neural network Lyapunov functions for stability analysis, where the trained Lyapunov functions were not verified. Here, we use the example to illustrate the compositional verification functionalities of LyZNet.

For the compositional approach, we examine two decompositions. The first splits the system into five linear subsystems, each with two variables. The nonlinear terms are treated as interconnections. The second has ten linear subsystems of the form $\dot{x}_i = -x_i$, each with one variable, and considers the remaining terms as interconnections. LyZNet automates these decompositions.

We tried verification over different domains with different approaches. The results are summarized in Table 3. It can be seen that for a smaller domain $X = [-1, 1]^{10}$, all verifiers work well and find the largest possible sublevel sets contained in the domain. It is worth noting that only the decomposition "10×1 dim" gives a verified ROA equal to the entire domain (barring the conservativeness of the numerical SMT verifier dReal [13]), because the other two approaches either produce the largest Euclidean ball or the largest Cartesian product of Euclidean balls contained in the domain.

As the domain gets larger, it becomes more challenging for the monolith approach. For $X = [-10, 10]^{10}$, the monolith approach takes more than 12 hrs without returning a result, while all compositional approaches succeeded. Notably, the "10×1 dim" decomposition provides the largest possible hyper-rectangular invariance set within the domain; indeed, it can be verified this set is $[-5, 5]^{10}$.

We also trained neural network functions and observe that, if we restrict the structural complexity of the neural network, we are able to verify regions of attraction using neural Lyapunov functions. However, the region of attraction is not as good as the ones obtained using quadratic Lyapunov functions, as those ones are already optimal for this example.

6 CONCLUSIONS AND FUTURE WORK

We presented a lightweight Python framework for learning and verifying neural Lyapunov functions. We demonstrated that by solving Zubov's PDE using neural networks, the verified region of attraction can indeed approach the boundary of the domain of attraction, outperforming sums-of-squares Lyapunov functions obtained using semidefinite programming. To cope with learning and verifying Lyapunov functions for high-dimensional systems, we have built support for compositional verification and demonstrated its effectiveness over a monolithic approach. While not presented in this paper, the compositional approach has been demonstrated to

Table 3: Verification results for a 10-dimensional system using quadratic Lyapunov functions (Example 5.2)

Domain	Approach	Verifier	Levels	Time (sec)
$[-1,1]^{10}$	monolith	local	0.49999	0.19
	compositional (5×2 dim)	local	0.49999	0.12
	compositional (10×1 dim)	local	0.49999	0.29
$[-4,4]^{10}$	monolith	local	7.99999	4.30
	compositional (5×2 dim)	local	7.99999	0.12
	compositional (10×1 dim)	local	3.1210	4.45
	compositional (10×1 dim)	quadratic	7.99999	1.54
$[-10,10]^{10}$	monolith	local	–	time out (>43200 sec)
	compositional (5×2 dim)	local	12.4938	2.07
	compositional (10×1 dim)	local	3.1188	4.57
	compositional (10×1 dim)	quadratic	12.4969	23.40

be effective for learning and verifying neural Lyapunov functions [28].

There are numerous ways the tool can be expanded. Ongoing research focuses on compositional training and verification of neural Lyapunov functions [28]. Future work could include supporting verification engines other than dReal, such as Z3 [10], and leveraging the growing literature on neural network verification tools. Extending support to other types of dynamical systems like delay differential equations and stochastic dynamics is also of interest. Exploring other stability and boundedness notions, such as global stability and ultimate boundedness, could be valuable. For compositional verification, designing neural networks that automatically discover compositional structures in high-dimensional systems is a promising direction, as opposed to the current manual assignment. Integrating convex optimization and semidefinite programming for suitable problems is another avenue. Data-driven computation of verifiable Lyapunov functions using Zubov's equation is another interesting direction [32, 33]. Finally, the tool has the potential to handle controls, making the simultaneous training of controllers and Lyapunov/value functions a natural next step. Initial results have been reported in [34] and to appear in [29].

ACKNOWLEDGMENTS

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada and the Canada Research Chairs program. The development of the tool was also enabled in part by support provided by the Digital Research Alliance of Canada (alliance.ca).

REFERENCES

[1] Alessandro Abate, Daniele Ahmed, Alec Edwards, Mirco Giacobbe, and Andrea Peruffo. 2021. FOSSIL: a software tool for the formal synthesis of Lyapunov functions and barrier certificates using neural networks. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*. 1–11.
[2] Alessandro Abate, Daniele Ahmed, Mirco Giacobbe, and Andrea Peruffo. 2020. Formal synthesis of Lyapunov neural networks. *IEEE Control Systems Letters* 5, 3 (2020), 773–778.
[3] Daniele Ahmed, Andrea Peruffo, and Alessandro Abate. 2020. Automated and sound synthesis of Lyapunov functions with SMT solvers. In *Proc. of TACAS*. Springer, 97–114.
[4] Ya-Chien Chang, Nima Roohi, and Sicun Gao. 2019. Neural Lyapunov control. *Advances in Neural Information Processing Systems* 32 (2019).
[5] Shaoru Chen, Mahyar Fazlyab, Manfred Morari, George J Pappas, and Victor M Preciado. 2021. Learning Lyapunov functions for hybrid systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*. 1–11.
[6] Shaoru Chen, Mahyar Fazlyab, Manfred Morari, George J Pappas, and Victor M Preciado. 2021. Learning region of attraction for nonlinear systems. In *Proceedings of the IEEE Conference on Decision and Control*. IEEE, 6477–6484.
[7] Hongkai Dai, Benoit Landry, Marco Pavone, and Russ Tedrake. 2020. Counter-example guided synthesis of neural network Lyapunov functions for piecewise linear systems. In *Proc. of CDC*. 1274–1281.
[8] Hongkai Dai, Benoit Landry, Lujie Yang, Marco Pavone, and Russ Tedrake. 2021. Lyapunov-stable neural-network control. In *Proc. of RSS*.
[9] Charles Dawson, Sicun Gao, and Chuchu Fan. 2023. Safe control with learned certificates: A survey of neural Lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics* (2023).
[10] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proc. of TACAS*. Springer, 337–340.
[11] Alec Edwards, Andrea Peruffo, and Alessandro Abate. 2023. Fossil 2.0: Formal Certificate Synthesis for the Verification and Control of Dynamical Models. *arXiv preprint arXiv:2311.09793* (2023).
[12] Nathan Gaby, Fumin Zhang, and Xiaojing Ye. 2022. Lyapunov-Net: A deep neural network architecture for Lyapunov function approximation. In *Proceedings of the IEEE Conference on Decision and Control*. IEEE, 2091–2096.
[13] Sicun Gao, Soonho Kong, and Edmund M Clarke. 2013. dReal: an SMT solver for nonlinear theories over the reals. In *Proceedings of International Conference on Automated Deduction*. 208–214.
[14] Peter Giesl. 2007. *Construction of Global Lyapunov Functions Using Radial Basis Functions*. Vol. 1904. Springer.
[15] Peter Giesl and Sigurdur Hafstein. 2015. Review on computational methods for Lyapunov functions. *Discrete & Continuous Dynamical Systems-B* 20, 8 (2015), 2291.
[16] Lars Grüne. 2021. Computing Lyapunov functions using deep neural networks. *Journal of Computational Dynamics* 8, 2 (2021).
[17] Lars Grüne. 2021. Overcoming the curse of dimensionality for approximating Lyapunov functions with deep neural networks under a small-gain condition. *IFAC-PapersOnLine* 54, 9 (2021), 317–322.
[18] Wassim M Haddad and VijaySekhar Chellaboina. 2008. *Nonlinear Dynamical Systems and Control: A Lyapunov-based Approach*. Princeton University Press.
[19] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. 2020. Array programming with NumPy. *Nature* 585, 7825 (2020), 357–362.
[20] Joblib Development Team. 2023. *Joblib: running Python functions as pipeline jobs*. https://joblib.readthedocs.io/
[21] Morgan Jones and Matthew M Peet. 2021. Converse Lyapunov functions and converging inner approximations to maximal regions of attraction of nonlinear systems. In *Proc. of CDC*. IEEE, 5312–5319.
[22] Wei Kang, Kai Sun, and Liang Xu. 2023. Data-Driven Computational Methods for the Domain of Attraction and Zubov's Equation. *IEEE Trans. Automat. Control* (2023).
[23] James Kapinski, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan, and Nikos Arechiga. 2014. Simulation-guided Lyapunov analysis for hybrid dynamical systems. In *Proc. of HSCC*. 133–142.
[24] Larissa Khodadadi, Behzad Samadi, and Hamid Khaloozadeh. 2014. Estimation of region of attraction for polynomial nonlinear systems: A numerical method. *ISA Transactions* 53, 1 (2014), 25–32.
[25] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. 1998. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* 9, 5 (1998), 987–1000.
[26] Jun Liu, Yiming Meng, Maxwell Fitzsimmons, and Ruikun Zhou. 2023. Physics-Informed Neural Network Lyapunov Functions: PDE Characterization, Learning, and Verification. *arXiv preprint arXiv:2312.09131* (2023).
[27] Jun Liu, Yiming Meng, Maxwell Fitzsimmons, and Ruikun Zhou. 2023. Towards Learning and Verifying Maximal Neural Lyapunov Functions. In *Proceedings of IEEE Conference on Decision and Control*.

[28] Jun Liu, Yiming Meng, Maxwell Fitzsimmons, and Ruikun Zhou. 2024. Compositionally Verifiable Vector Neural Lyapunov Functions for Stability Analysis of Interconnected Nonlinear Systems. In *Proc. of ACC*.
[29] Jun Liu, Yiming Meng, and Ruikun Zhou. 2024. LyZNet with Control: Physics-Informed Neural Network Control of Nonlinear Systems with Formal Guarantees. In *Proceedings of the 8th IFAC Conference on Analysis and Design of Hybrid Systems*.
[30] Y. Long and M.M. Bayoumi. 1993. Feedback stabilization: control Lyapunov functions modelled by neural networks. In *Proc. of CDC*. 2812–2814 vol.3.
[31] Aleksandr Mikhailovich Lyapunov. 1992. The general problem of the stability of motion. *Internat. J. Control* 55, 3 (1992), 531–534.
[32] Yiming Meng, Ruikun Zhou, and Jun Liu. 2023. Learning Regions of Attraction in Unknown Dynamical Systems via Zubov-Koopman Lifting: Regularities and Convergence. *arXiv preprint arXiv:2311.15119* (2023).
[33] Yiming Meng, Ruikun Zhou, and Jun Liu. 2024. Zubov-Koopman Learning of Maximal Lyapunov Functions. In *Proceedings of American Control Conference*.
[34] Yiming Meng, Ruikun Zhou, Amartya Mukherjee, Maxwell Fitzsimmons, Christopher Song, and Jun Liu. 2024. Physics-Informed Neural Network Policy Iteration: Algorithms, Convergence, and Verification. *arXiv preprint* (2024).
[35] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. 2017. SymPy: symbolic computing in Python. *PeerJ Computer Science* 3 (2017), e103.
[36] Andy Packard, Ufuk Topcu, Peter J Seiler Jr, and Gary Balas. 2010. Help on SOS. *IEEE Control Systems Magazine* 30, 4 (2010), 18–23.
[37] Antonis Papachristodoulou and Stephen Prajna. 2002. On the construction of Lyapunov functions using the sum of squares decomposition. In *Proceedings of IEEE Conference on Decision and Control*, Vol. 3. IEEE, 3482–3487.
[38] Antonis Papachristodoulou and Stephen Prajna. 2005. A tutorial on sum of squares techniques for systems analysis. In *Proc. of ACC*. IEEE, 2686–2700.
[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (2019).
[40] Danil V Prokhorov. 1994. A Lyapunov machine for stability analysis of nonlinear systems. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, Vol. 2. IEEE, 1028–1031.
[41] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics* 378 (2019), 686–707.
[42] Rodolphe Sepulchre, Mrdjan Jankovic, and Petar V Kokotovic. 2012. *Constructive Nonlinear Control*. Springer Science & Business Media.
[43] Weehong Tan and Andrew Packard. 2008. Stability region analysis using polynomial and composite polynomial Lyapunov functions and sum-of-squares programming. *IEEE Trans. Automat. Control* 53, 2 (2008), 565–571.
[44] Ufuk Topcu, Andrew Packard, and Peter Seiler. 2008. Local stability analysis using simulations and sum-of-squares programming. *Automatica* 44, 10 (2008), 2669–2675.
[45] Anthony Vannelli and Mathukumalli Vidyasagar. 1985. Maximal Lyapunov functions and domains of attraction for autonomous nonlinear systems. *Automatica* 21, 1 (1985), 69–80.
[46] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* 17, 3 (2020), 261–272.
[47] Ruikun Zhou, Thanin Quartz, Hans De Sterck, and Jun Liu. 2022. Neural Lyapunov Control of Unknown Nonlinear Systems with Stability Guarantees. In *Advances in Neural Information Processing Systems*.
[48] V. I. Zubov. 1964. *Methods of A. M. Lyapunov and Their Application*. Noordhoff.

7 APPENDIX

This appendix presents the dynamic equations for the numerical examples discussed in Section 5. Additionally, we report the computational times required for the training and verification processes in Examples 5.1 and 5.2.

7.1 Van der Pol equation

The reversed Van der Pol equation is given by

$$\begin{aligned} \dot{x}_1 &= -x_2, \\ \dot{x}_2 &= x_1 - \mu(1 - x_1^2)x_2, \end{aligned} \tag{20}$$

where $\mu > 0$ is a parameter that affects the stiffness of the equation.

7.2 Two-machine power system

Consider the two-machine power system [45] modelled by

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= -0.5x_2 - (\sin(x_1 + \delta) - \sin(\delta)), \end{aligned} \tag{21}$$

where $\delta = \frac{\pi}{3}$.

7.3 Inverted pendulum with linear control

Consider an inverted pendulum

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= \sin(x_1) - x_2 - (k_1 x_1 + k_2 x_2), \end{aligned} \tag{22}$$

where the linear gains are given by $k_1 = [4.4142, 2.3163]$.

7.4 Synthetic 10-dimensional system

Consider the 10-dimensional nonlinear system from [16]:

$$\begin{aligned} \dot{x}_1 &= -x_1 + 0.5x_2 - 0.1x_9^2, \\ \dot{x}_2 &= -0.5x_1 - x_2, \\ \dot{x}_3 &= -x_3 + 0.5x_4 - 0.1x_1^2, \\ \dot{x}_4 &= -0.5x_3 - x_4, \\ \dot{x}_5 &= -x_5 + 0.5x_6 + 0.1x_7^2, \\ \dot{x}_6 &= -0.5x_5 - x_6, \\ \dot{x}_7 &= -x_7 + 0.5x_8, \\ \dot{x}_8 &= -0.5x_7 - x_8, \\ \dot{x}_9 &= -x_9 + 0.5x_{10}, \\ \dot{x}_{10} &= -0.5x_9 - x_{10} + 0.1x_2^2. \end{aligned}$$

7.5 Computational times

Table 4: Training and verification times for Examples 5.1 (Van der Pol) and 5.2 (two-machine power)

Model	Training (sec)	Verification (sec)
Van der Pol ($\mu = 1$)	494	973
Van der Pol ($\mu = 3$)	502	442
two-machine power	473	3375

We report the computational times required for the training and verification processes in Examples 5.1 and 5.2 on a 2020 MacBook Pro with a 2 GHz Quad-Core Intel Core i5, without any GPU. For training, we randomly chose 300,000 collocation points in the domain and trained for 20 epochs. The sizes of the neural networks were as reported in Tables 1 and 2. Verification in dReal can be easily parallelized by adjusting the config parameter `number_of_jobs`, which may correspond to the number of cores/threads on the computer. For the laptop used in these experiments, there are four cores, allowing for eight threads via hyperthreading. As discussed in Remark 3, computational time is largely dependent on computer hardware. The computational times provided in Table 4 serve as a reference point. We expect that training time can be significantly improved by using GPUs, and verification time can be further reduced by utilizing additional cores.