Accelerating Lyapunov-Stable Neural Control using Fulfillment Priority Logic

Abdelrahman Abdelgawad[1], Bassel El Mabsout[2], Zili Wang[1], Renato Mancuso[2], Sean B. Andersson[1,3], and Roberto Tron[1,3]

***Abstract*— We present a two-stage approach for learning stability-certified neural controllers that achieves a reduction of up to $\sim 95\%$ in training time compared to the state-of-the-art baseline, which introduced monotonic neural Lyapunov architectures. Our method combines monotonic neural Lyapunov functions with fulfillment priority logic (FPL) to efficiently initialize controllers before formal verification.**

Traditional approaches for jointly learning controllers and neural Lyapunov functions require computationally expensive mixed-integer linear programming (MILP) or satisfiability modulo theory (SMT) solvers at each training iteration, often taking several hours to converge. We address this bottleneck by leveraging FPL to perform early joint initialization of the controller and Lyapunov networks. Building on the monotonic neural network architecture from the baseline, which guarantees non-negativity and a unique global minimum by construction, our method focuses on efficiently satisfying the remaining property of decreasing along trajectories.

Existing works focus on maximizing the region of attraction/convergence of the learned controller. In contrast, leveraging FPL allows us to (1) increase learning efficiency substantially and (2) focus on complementary performance metrics, such as convergence rate and control effort minimization, thereby adding significant specification flexibility.

In this paper, we encode an approximate Lyapunov-decrease condition in FPL to pre-train the controller and Lyapunov networks, then apply a MILP-based verification/refinement step. This decouples efficient learning from certificate enforcement and allows the FPL specification to include auxiliary objectives (e.g., convergence rate and control effort), whose influence persists through the final MILP pass. The resulting controllers converge rapidly while admitting formal Lyapunov certificates on standard nonlinear control benchmarks.[1]

I. Introduction

Ensuring the stability of learned controllers in nonlinear systems is a central challenge in safety-critical systems. Control Lyapunov functions (CLFs) provide a principled route to certified stabilization: if a function V satisfies positivity and is guaranteed to decrease along closed-loop trajectories on a domain $\mathcal{X}$, convergence to the equilibrium follows [1]. However, constructing suitable CLFs and synthesizing the accompanying controllers remain challenging for practical systems. Linearization-based designs (e.g., LQR) offer only local guarantees [1] and can be overly conservative [2]. While sum-of-squares (SOS) methods extend beyond local analysis by using semi-definite programming (SDP) to find polynomial Lyapunov functions [3], [4], they suffer from restricted polynomial expressivity [5] and poor computational scaling with system dimension [6].

Recently, a line of work has emerged that attempts to address the limitations of the aforementioned methods by utilizing *neural network* Lyapunov function candidates. Despite the universal approximation capability of neural networks, enforcing Lyapunov conditions on these parameterizations while maintaining tractability remains a fundamental challenge that has sparked a wealth of verification and synthesis approaches. The work in [6] introduced neural Lyapunov learning as a counterexample-guided loop alternating between optimization and formal falsification using delta-complete SMT solvers such as dReal, demonstrating enlarged verified regions of attraction (ROAs) on robotics benchmarks. Subsequent efforts switched to ReLU-based architectures, whose piecewise-affine structure maps exactly to a MILP—one binary variable and two linear constraints per neuron—allowing the worst-case Lyapunov-decrease violation to be found to global optimality without the numerical relaxation inherent in delta-complete SMT solving. This exact counterexample oracle was then embedded in a bilevel training loop, enabling joint synthesis of neural controllers and Lyapunov functions for nonlinear systems [7], [8]. Recent advances in neural Lyapunov function learning have focused on architectures that guarantee positive definiteness by construction; notably, monotonic Lyapunov networks enforce non-negativity and a unique global minimum by design, so only the decrease condition requires verification [9]. However, the remaining verification still relies on solving a MILP at every training iteration to find the maximum Lyapunov-decrease violation and then differentiating through the resulting active set to update network parameters. This MILP-in-the-loop procedure remains the dominant computational bottleneck, making even moderately sized systems (e.g., a 3D quadrotor) take hours to train.

A complementary line of work learns control certificates—including neural CLFs/CBFs and Lyapunov–Barrier hybrids—through sampling-based optimization rather than exact solver-in-the-loop verification. In some methods, these certificates are enforced online through lightweight constrained optimizations, such as CLF/CBF-QPs or related hybrid controllers, to preserve safety and task progress at runtime. Other methods jointly learn controllers and certificates directly, emphasizing scalability, generalization, or unknown-dynamics settings rather than exact end-to-end formal verification. Together,

[1]Division of Systems Engineering, [2]Dept. of Computer Science, [3]Dept. of Mechanical Engineering, Boston University, Boston, MA 02215, USA. {aaoaa,bmabsout,zw2445,rmancuso,sanderss,tron}@bu.edu.

This work was supported in part by NSF FRR-2212051.

[1]Code available at the following GitHub repository https://github.com/AbdelrahmanAbdelgwad/fpl-neural-lyapunov.

these works highlight strong practical promise for certificate-based learning, while also stressing the importance of reducing the computational burden of certification [4], [10]–[12].

Despite these advances, two practical obstacles persist. (1) *Computational burden*: many methods interleave solver calls with training, repeatedly invoking MILP/SMT to find counterexamples—an expensive loop that dominates wall-clock time. (2) *Need for good initialization*: without a good warm-start for both the controller and the Lyapunov candidate, optimization can settle in poor local minima or require careful initialization, e.g., using linearization and LQR.

Fulfillment Priority Logic (FPL), introduced in [13], has demonstrated strong success in improving learning sample efficiency in reinforcement learning. The framework allows composing multiple (competing) objectives while preserving their logical relationships, achieving up to 500% improvement in sample efficiency over standard methods such as Soft Actor-Critic [13]. FPL models each design goal as a fulfillment variable $f \in [0, 1]$ and composes goals using the power-mean operator, which smoothly interpolates between conservative (AND-like) and optimistic (OR-like) behavior.

Our approach. We employ a monotonic Lyapunov parameterization with a two-term FPL composition: (i) a Lyapunov-*progress* surrogate that encourages a fixed per-step decrease over sampled rollouts, and (ii) a task term encoding system-specific performance objectives. Our method operates in two stages. In Stage I, sampling-based FPL approximates the Lyapunov conditions without expensive verification, leaving minimal repairs for the Stage II MILP solver.
Our contributions are as follows:

- A sampling-based FPL approximation of the Lyapunov conditions over trajectory rollouts for joint controller-Lyapunov initialization, reducing costly MILP certify–repair iterations.
- Simulations on nonlinear control benchmarks (pendulum and unicycle) showing up to $\sim 95\%$ wall-clock time reduction while retaining formal certificates.
- The ability to encode task-relevant objectives (e.g., convergence rate and control effort) alongside stability, persisting through MILP verification.

Organization. Section II presents the stability certification problem and reviews monotonic neural Lyapunov functions and FPL fundamentals. Section III presents our two-stage method: an FPL-based approximate solver followed by exact MILP verification. Section IV demonstrates our approach on pendulum and unicycle systems. Section V concludes and discusses future work.

II. Problem Statement

A. Problem Setup

We consider a discrete-time piecewise linear system

$$x_{k+1} = f(x_k, u_k), \qquad u_k = \pi_\theta(x_k), \tag{1}$$

on a bounded domain $\mathcal{X} \subset \mathbb{R}^{n_x}$ with constrained input $u_k \in \mathcal{U} = \{u \in \mathbb{R}^{n_u} \mid \|u\|_\infty \le u_{\max}\}$. We assume that the system has an equilibrium at $(x_{\rm eq}, u_{\rm eq})$ satisfying $x_{\rm eq} = f(x_{\rm eq}, u_{\rm eq})$ and that the system is locally exponentially stabilizable.

We seek a feedback control policy $\pi_\theta : \mathcal{X} \to \mathcal{U}$ and a control Lyapunov function (CLF) $V : \mathcal{X} \to \mathbb{R}$ that guarantees exponential stability [1]. Specifically, we require that for some region $\mathcal{D} \subseteq \mathcal{X}$ and decay rate $\epsilon > 0$:

$$V(x_{\rm eq}) = 0, \tag{2a}$$
$$V(x) > 0, \quad \forall x \in \mathcal{D} \setminus \{x_{\rm eq}\}, \tag{2b}$$
$$V(f(x, \pi_\theta(x))) - V(x) \le -\epsilon V(x)\ , \forall x \in \mathcal{D} \setminus \{x_{\rm eq}\}. \tag{2c}$$

These conditions ensure that V is positive definite with respect to $x_{\rm eq}$ and decreases along system trajectories under the control policy π_θ. If V is radially unbounded, like in the case of monotonic networks [9], and $\mathcal{D} = \mathcal{X}$, the stability is global.

B. Problem Formulation and Design Goals

Synthesis with Fixed Domain: We consider the problem of jointly learning a piecewise linear neural network controller $\pi_\theta : \mathcal{X} \to \mathcal{U}$ and a piecewise linear neural (monotonic) Lyapunov function $V_\psi : \mathcal{X} \to \mathbb{R}$ for the piecewise linear system (1) and seek to achieve the following goals.

Goal 1 (Stability): The learned controller must stabilize the system with formal guarantees. This requires finding V_ψ and π_θ such that the Lyapunov conditions in (2) are satisfied over the region of interest.

Goal 2 (Computational Efficiency): The learning process must be computationally tractable, minimizing expensive per-iteration MILP solver calls.

Goal 3 (Performance): Beyond stability, the controller should optimize a desired control objective—such as low control effort or fast convergence to the equilibrium—without compromising stability.

C. Monotonic Neural Lyapunov Functions

Monotonic neural networks, introduced in [9], guarantee positive definiteness by construction through layers that are monotonically increasing along radial directions from the equilibrium. A monotonic layer $M : \mathbb{R}^{d_{\rm in}} \to \mathbb{R}^{d_{\rm out}}$ is formed by stacking $d_{\rm out}$ scalar components, where the j-th component is:

$$[M]_j(x) = \sum_{i=1}^{n_M} c_i\, m_i(v_i^\top x),$$

where each unit $m_i(y) = a_i^\top \mathrm{ReLU}(y\mathbf{1} - b_i)$ is piecewise linear with $a_i, b_i \in \mathbb{R}^p$ (p being the number of linear pieces per unit), $v_i \in \mathbb{R}^{d_{\rm in}}$ are fixed direction vectors whose convex hull contains the origin, $c_i > 0$, and $n_M \ge d_{\rm in} + 1$. The parameters a_i and b_i are constrained to ensure monotonicity along each direction v_i. In general, a monotonic Lyapunov network ϕ_v is defined as a composition of monotonic layers M_k. That is, $\phi_v = M_n \circ M_{n-1} \circ \cdots \circ M_1 : \mathbb{R}^{n_x} \to \mathbb{R}$, with the input dimension of the first monotonic layer $d_{in}^1 = n_x$ and the output dimension of the last layer $d_{out}^n = 1$. In this work we use a single monotonic layer with the number of direction vectors v_i and pieces p dependent on the example ($\phi_v = M_1 : \mathbb{R}^{n_x} \to \mathbb{R}$). See Table I for details.

The Lyapunov candidate is defined as

$$V_\psi(x) = \phi_v(x) - \phi_v(x_{eq}) + \lambda \|R(x - x_{eq})\|_1, \quad (3)$$

where $\lambda > 0$ and $R \in \mathbb{R}^{m \times n_x}, m \geq n_x$, is full rank. This architecture automatically satisfies conditions (2a) and (2b), guaranteeing $V(x_{eq}) = 0$ and positive definiteness by construction with a unique global minimum at the equilibrium. All networks in this work use Leaky ReLU activations, defined as $\sigma(y) = \max(cy, y)$ with $0 \leq c < 1$. These reduce to a standard ReLU when $c = 0$.

We verify the decrease condition (2c) by iteratively solving a MILP. The details of this approach can be found in [9]; below we include a brief summary. Condition (2c) is encoded as the maximization problem

$$\zeta^* = \max_{x \in \mathcal{D}} \left(V_\psi(f(x, \pi_\theta(x))) - (1 - \epsilon) V_\psi(x) \right), \quad (4)$$

whose piecewise-affine structure follows from the use of Leaky ReLU activations in V_ψ and $f \circ \pi_\theta$. Each Leaky ReLU unit $\sigma(y) = \max(cy, y)$ is represented exactly using binary variables $\beta \in \{0, 1\}$ and linear constraints that select the active branch. If $\zeta^* \leq 0$, then stability is certified over $\mathcal{D}$; otherwise, the MILP solution provides a violating state used to update the network parameters.

Following [9] and [8], after solving the MILP to optimality, we fix the optimal binary activation pattern and the active linear constraints, yielding a locally affine representation of the optimal objective ζ^* as a function of the network parameters. We then differentiate this active-set surrogate and use the resulting gradient to update the parameters:

$$\Theta = \{\theta\} \cup \{\psi\} \cup \{R\}, \quad (5a)$$

$$\Theta \leftarrow \Theta - \alpha \frac{\partial \zeta^*}{\partial \Theta}. \quad (5b)$$

Here, θ and ψ denote the neural network parameters of π_θ and V_ψ, respectively, and α is the learning rate.

D. Fulfillment Priority Logic

Fulfillment Priority Logic (FPL), introduced in [13], is a specification language for differentiable multi-objective composition. FPL maps the i-th objective to a *fulfillment* value $f_i \in [0, 1]$ and composes the fulfillment values via the power mean:

$$\left(\frac{1}{n} \sum_{i=1}^{n} f_i^p \right)^{1/p}, \quad p \in \mathbb{R}, \quad (6)$$

which interpolates between $\min_i f_i$ ($p \to -\infty$, AND-like) and $\max_i f_i$ ($p \to \infty$, OR-like). Choosing $p \leq 0$ yields the conjunction operator $\wedge_p$, ensuring all objectives must be highly fulfilled for the composition to be high.

Unlike a weighted sum, FPL normalization to $[0, 1]$ removes the scale dependence between differently unitized signals (e.g., Lyapunov decrease vs. control effort), and the power mean enables principled logical composition without extensive manual weight tuning.

We leverage FPL to formulate a sampling-based approximation of the Lyapunov conditions that guides joint controller-Lyapunov learning. For our problem, we compose two fulfillments: **(i)** a *stability* fulfillment f_{stab} that encourages the Lyapunov function to decrease along trajectory rollouts, approximating (2c); and **(ii)** a *task* fulfillment f_{task} encoding objectives such as convergence speed and control effort. The composed objective is:

$$f_{\text{total}} = f_{\text{stab}} \wedge_p f_{\text{task}} = \left(\frac{1}{2} (f_{\text{stab}}^p + f_{\text{task}}^p) \right)^{1/p}. \quad (7)$$

This formulation enables gradient-based optimization of both π_θ and V_ψ via backpropagation, avoiding repeated MILP calls during initial training. The specific definitions of f_{stab} and f_{task} are given in Section III.

III. Methodology

Our approach addresses the three design goals mentioned in Section II-B through a two-stage process: (i) In Stage I we sample trajectories, compute the fulfillment value for the Lyapunov decrease condition and task-based objectives, and take gradient steps until we find a high-fulfillment controller-Lyapunov pair without verification; (ii) in Stage II, exactly like the baseline [9], we use a MILP-based certify-repair loop that provides formal guarantees on convergence.

A. From objectives to an FPL specification

We work with two types of signals: (i) a *stability-progress* signal that reflects the decrease in the Lyapunov candidate along a rollout trajectory, and (ii) a *task* signal that encodes proximity to the desired behavior. We map each scalar score z to a fulfillment in $[0, 1]$ using a *monotone piecewise–linear* squashing function with calibrated breakpoints. This allows for meaningful composition with FPL and yields robust gradients by saturating outliers.

Formally, let $\mathcal{P} = \{(\tau_0, y_0), \ldots, (\tau_m, y_m)\}$ with $\tau_0 < \cdots < \tau_m$ and $y_0, \ldots, y_m \in [0, 1]$. Define

$$g_{\text{pw}}(z; \mathcal{P}) = \begin{cases} y_0, & z \leq \tau_0, \\ y_j + \dfrac{y_{j+1} - y_j}{\tau_{j+1} - \tau_j} (z - \tau_j), & \tau_j < z \leq \tau_{j+1}, \\ y_m, & z > \tau_m, \end{cases} \quad (8)$$

which maps to $[0, 1]$ and is monotone provided the sequence $y_0, \ldots, y_m$ is itself monotone (non-decreasing or non-increasing).

For each initial state i in the minibatch $\mathcal{B}$, we roll out a trajectory of horizon H steps. We define the Lyapunov drop $\Delta V = V_\psi(x_0) - V_\psi(x_H)$. We target a per-step decrease of at least κ, making our *per-horizon* target

$$T^{(i)} = \min\{\kappa H, V_\psi(x_0^{(i)})\}, \qquad \kappa = 1/N,$$

where N is the desired convergence horizon: the number of steps in which the controller is expected to drive the Lyapunov value to zero. The target decrease is limited from above by the current Lyapunov value, as it is the maximum possible decrease that the Lyapunov value can take. A controller would be considered to decrease the value over the horizon H "fast enough" if $\Delta V \geq T$. We then build the stability fulfillment

by squashing $\Delta V^{(i)}$ and aggregating across the minibatch dimension:

$$f_{\text{stab}} \;=\; \wedge_{-2}\Big\{\, g_{\text{pw}}\big(\Delta V^{(i)};\,(-1,0),\,(-0.1,10^{-3}),\,(0,10^{-2}),\,(T^{(i)},0.9),\,(1,1)\big)\Big\}_{i\in\mathcal{B}}. \tag{9}$$

Notice that $\Delta V \le 0$ is mapped to a fulfillment of $\le 10^{-2}$, while matching the schedule $\Delta V = T$ yields a high score (0.9), and additional progress smoothly saturates to 1. Note that in all experiments below we use N=100 (thus κ=0.01).

Task-specific calibration: In the numerical experiments in Sec. IV we utilize two example systems: (1) **Inverted pendulum** and (2) **Path-following unicycle**. For both systems, we use the *same* dynamics as in [9]: the second-order inverted pendulum (System I, Eq. (26) in [9]) and the curvature-dependent unicycle on the unit circle (System II, Eq. (27) in [9]). We refer the reader to [9] for the explicit forms and domains. Here we give the specific FPL tasks used for each system.

Pendulum. Let $s(\theta_t) \in [0,1]$ be the normalized angular distance (1 at upright, 0 at opposite). We aggressively aggregate the fulfillment value across both time and minibatch dimensions simultaneously:

$$z^{(\text{pend})}_{\text{angle}} \;=\; \wedge_{-2}\big\{\, s(\theta_t^{(i)})\,\big\}_{t=0,i\in\mathcal{B}}^{H}, \tag{10}$$

then remap via

$$f^{(\text{pend})}_{\text{angle}} \;=\; g_{\text{pw}}\Big(z^{(\text{pend})}_{\text{angle}};\,(0,0),\,(0.6,10^{-2}),\,(0.7,0.9),\,(1,1)\Big). \tag{11}$$

We also minimize control effort. Let $e_t = \|u_t\|_2$. We squash each e_t via $g_{\text{pw}}(e_t^{(i)};\mathcal{P}_{\text{effort}})$, then aggregate over time with $p=-2$ and across the minibatch with $p=1$:

$$f^{(\text{pend})}_{\text{effort}} \;=\; \wedge_{1}\Big\{\; \wedge_{-2}\Big\{\, g_{\text{pw}}\big(e_t^{(i)}\big)\,\Big\}_{t=0}^{H}\Big\}_{i\in\mathcal{B}}, \tag{12}$$

where $\mathcal{P}_{\text{effort}} = \{(0,1),\,(5,0.9),\,(8,0.5),\,(15,0.1),\,(20,0)\}$. Finally, the pendulum task fulfillment composes angle and effort with a balanced conjunction ($p=0$):

$$f^{(\text{pend})}_{\text{task}} \;=\; f^{(\text{pend})}_{\text{angle}} \wedge_0 f^{(\text{pend})}_{\text{effort}}. \tag{13}$$

Unicycle. We use a proximity-to-equilibrium signal $r^{(i)}$ (smaller is better) that measures how close the final state is to the equilibrium compared to the expected given the horizon. We squash it with a *decreasing* calibration and aggregate across the minibatch with $p=-2$:

$$f^{(\text{uni})}_{\text{task}} \;=\; \wedge_{-2}\Big\{\, g_{\text{pw}}\big(r^{(i)};\,(-10,1),\,(-0.1,0.9),\,(0,10^{-2}),\,(0.1,10^{-3}),\,(1,0)\big)\Big\}_{i\in\mathcal{B}}. \tag{14}$$

The breakpoints in Eqs. (9)–(14) were minimally hand-tuned to obtain the best performance for each system. By design, FPL assumes domain knowledge to define fulfillment mappings that meaningfully normalize raw signals to $[0,1]$.

TABLE I: Hidden layer sizes of NNs [9]. Brackets: hidden layers; dirs: direction vectors; pcs: pieces per unit.

	Dynamics f	Controller π_θ	Lyapunov V_ψ
Pendulum	[5, 5]	[3, 2]	5 dirs / 4 pcs
Unicycle	[6, 6]	[4, 3]	6 dirs / 4 pcs

FPL composition: The total objective composes the two aggregated fulfillments with a balanced conjunction:

$$f_{\text{total}} \;=\; f_{\text{stab}} \;\wedge_0\; f^{(\text{system})}_{\text{task}}, \tag{15}$$

where f_{stab} is defined in Eq. (9) and $f^{(\text{system})}_{\text{task}}$ is the system-specific task fulfillment defined in Eqs. (13) and (14).

B. Stage I: FPL-guided joint pre-training

We roll out the closed loop $x_{k+1} = f(x_k, \pi_\theta(x_k))$ from initial states sampled as $x_0 \sim \mathcal{N}(x_{\text{eq}}, \text{diag}((x_{\text{up}} - x_{\text{lo}})^2))$, clamped to $\mathcal{D}$. The horizon H, for both systems, is sampled uniformly from a short range of $\{1,\dots,15\}$ to encourage a consistent decrease rather than one-step greediness. We terminate Stage I after a max number of epochs (200 for pendulum and 80 for unicycle) or when the epoch-average fulfillment exceeds 0.98.

We use Adaptive Moment Estimation (Adam) on the loss $\mathcal{L}_{\text{FPL}} = 1 - f_{\text{total}}$. This shapes the controller and V_ψ to (i) decrease V along rollout trajectories and (ii) satisfy task objectives, starting from randomly initialized weights.

C. Stage II: MILP verification

After attaining the approximate solution via Stage I, we initialize the certify-repair loop with the parameters Θ obtained from Stage I. The loop then works on verifying (2c) and updating the weights over $\mathcal{D}$ with an exact MILP obtained by encoding the monotonic-based V_ψ, the ReLU-based π_θ, and the ReLU-based dynamics into linear constraints with binary activations. The corresponding network architectures used in each experiment are summarized in Table I. For details, see [9].

IV. Numerical Experiments

A. Experimental Setup

Every experiment compares two configurations: (1) the baseline [9], which uses only Stage II (MILP-based verification), serving as an ablation of Stage I; and (2) our method (Algorithm 1), which prepends Stage I (FPL pre-training) before Stage II.

Hardware and software: All runs were performed on an AMD Ryzen Threadripper 3960x (24 cores/48 threads), 62.7 GiB RAM, Ubuntu 20.04.6 LTS. We note that all experiments were run on CPU only; no GPU was used. We use PyTorch for learning and Gurobi for MILP verification.

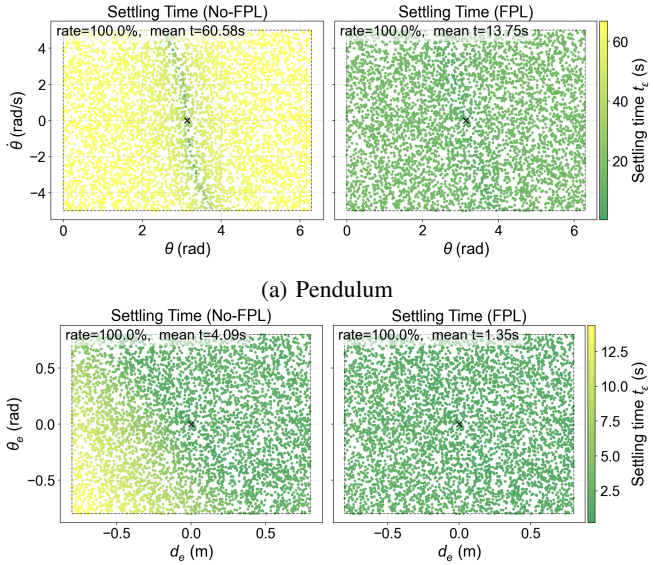
B. Training Protocol

Each experiment reports the mean wall-clock *training* time over 5 independent trials (fresh seeds and randomized minibatches). Timing is measured externally in the bash wrappers via `SECONDS` and printed at the end of each run. The same horizon H is used per minibatch $\mathcal{B}$.

Algorithm 1 FPL-guided monotonic Lyapunov learning with MILP verification

Require: Dynamics f; policy π_θ; Lyapunov net V_ψ; bounds $[x_{\text{lo}}, x_{\text{up}}]$; $\kappa{=}1/N$; $H \in [H_{\min}, H_{\max}]$; Adam

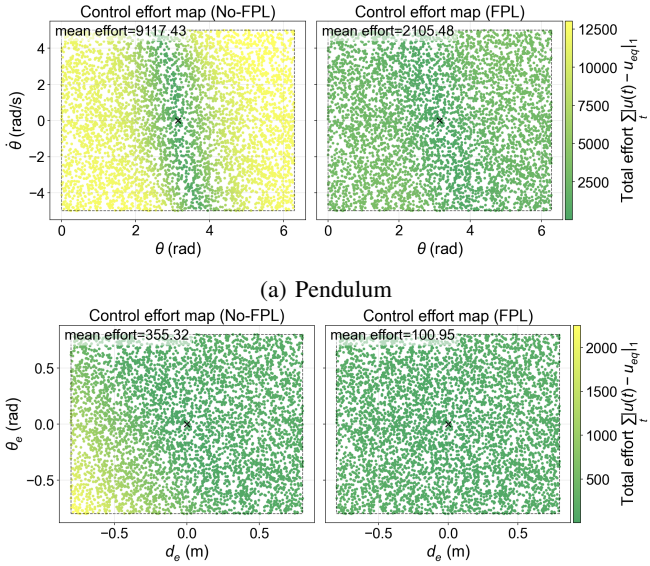
Stage I: FPL pre-training
for epochs **do**
  for each batch $\mathcal{B}$ **do**
    Sample $\{x_0^{(b)}\} \sim \mathcal{N}(x_{\text{eq}}, \text{diag}((x_{\text{up}} - x_{\text{lo}})^2))$
    Clamp $x_0^{(b)}$ to $[x_{\text{lo}}, x_{\text{up}}]$
    Sample $H \sim \text{Unif}[H_{\min}, \ldots, H_{\max}]$
    Roll out $x_{k+1} = f(x_k, \pi_\theta(x_k))$
    Compute $\Delta V^{(b)}$, $T^{(b)}$ for all b
    $f_{\text{stab}} \leftarrow \wedge_{-2}\{g_{\text{pw}}(\Delta V^{(b)}; T^{(b)})\}_{b \in \mathcal{B}}$ ▷ Eq. (9)
    $f_{\text{task}} \leftarrow$ system-specific score ▷ Eqs. (10)–(14)
    $f_{\text{total}} \leftarrow f_{\text{stab}} \wedge_0 f_{\text{task}}$
    Minimize $\mathcal{L}_{\text{FPL}}{=}1{-}f_{\text{total}}$ w.r.t. (θ, ψ, R)
  end for
  break if epoch-average fulfillment ≥ 0.98
end for
Stage II: MILP verification
Initialize Θ from Stage I
repeat
  Solve $\zeta^\star = \max_x V_\psi\big(f(x, \pi_\theta(x))\big) - V_\psi(x) + \epsilon V_\psi(x)$ ▷ Eq. (4)
  if $\zeta^\star > 0$ **then** Update (θ, ψ, R) via $\nabla_\Theta \zeta^\star$
  end if
until $\zeta^\star \leq 0$ ▷ certified



(a) Pendulum

(b) Path-following unicycle

Fig. 1: **Convergence time.** Each dot is one initial condition rolled out in the ReLU network dynamics model. Color encodes the settling time t_ε (16) using the *same* color limits across panels for a fair comparison. Faster convergence is **green**; slower transitions toward **yellow**. Dashed rectangle: evaluation domain; black "×": equilibrium.



(a) Pendulum

(b) Path-following unicycle

Fig. 2: **Control-effort.** Each dot is an initial condition; color encodes J_u (Eq. (17)) over the simulation horizon (same color limits across panels). Lower effort is **green**; higher effort is **yellow**. Dashed rectangles: evaluation domain; black "×": equilibrium.

TABLE II: Training time comparison (mean over 5 trials).

Environment	Without FPL	With FPL	Speedup
Inverted Pendulum	33m 56s	1m 48s	18.85×
Path-Following Unicycle	4m 1s	15 s	16×

C. Evaluation Metrics

For an initial condition x_0, we measure *settling time* to an ε-ball (with $\varepsilon = 10^{-3}$) and *control effort*:

$$t_\varepsilon(x_0) = \inf\{t \mid \|x(t; x_0) - x_{eq}\|_1 \leq \varepsilon\}, \tag{16}$$

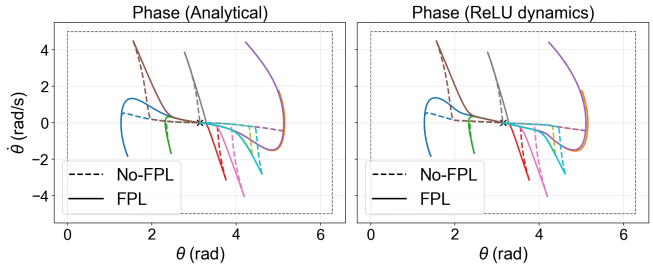
$$J_u(x_0) = \sum_{k=0}^{t_\varepsilon} \|u_k - u_{\text{eq}}\|_1. \tag{17}$$

Trials that do not satisfy (16) within a maximum horizon of 100 s (pendulum) and 50 s (unicycle) are considered non-convergent.

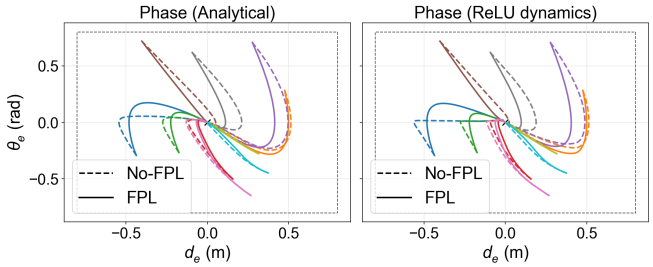
D. Results

a) Training time: Table II shows that prepending Stage I reduces wall-clock training time by 18.85× on the pendulum and 16× on the path-following unicycle. These speedups are achieved without sacrificing controller quality, as demonstrated by the metrics below.

b) Settling time: Figure 1 evaluates 5000 initial conditions under the learned dynamics. For the pendulum, the No-FPL (baseline) controller yields a mean settling time of ≈60.6 s, whereas FPL (our method) reduces this to ≈13.8 s (∼4.4× faster on average). For the unicycle, the mean drops from ≈4.09 s to ≈1.35 s (∼3.0× faster). Regions far from the equilibrium or with high state error contain the slowest points



(a) Pendulum



(b) Path-following unicycle

Fig. 3: **Phase diagrams.** Trajectories from several representative initial conditions. *Dashed* curves: controller trained **without** FPL; *solid* curves: **with** FPL. Pendulum plane $(\theta, \dot{\theta})$; unicycle plane (d_e, θ_e). Dashed rectangle: evaluation domain; black "×": equilibrium. Colors distinguish initial conditions.

for both controllers, yet the FPL panel is uniformly shifted toward the fast end of the scale (green), indicating broad improvements rather than isolated cases. Both controllers achieve 100% convergence over the displayed domain, so the color differences reflect purely faster transients under FPL rather than a trade-off with convergence rate.

c) Control effort: Figure 2 colors each initial condition by the cumulative control magnitude in Eq. (17). FPL substantially decreases the mean effort on both systems: from $\sim$9,117 to $\sim$2,105 on the pendulum ($\sim 4.3\times$ reduction) and from ~ 355 to ~ 101 on the unicycle ($\sim 3.5\times$). Figure 3 shows the trajectories under both controllers, revealing that the effort reduction reflects a qualitative change in closed-loop behavior rather than merely faster convergence: our controller follows fundamentally different paths that are more aligned with the encoded performance objectives.

V. Conclusion and Future Work

We presented a novel approach combining Fulfillment Priority Logic with monotonic neural Lyapunov functions to learn controllers with formally verified stability while dramatically reducing computational costs compared to an existing baseline. The two-stage pipeline—FPL-guided trajectory-based pre-training followed by MILP verification—achieved up to 95% reduction in training time compared to the baseline [9]. On benchmark tasks (pendulum and unicycle), our approach consistently produced controllers that pass formal verification in fewer MILP certify-repair iterations, demonstrating that FPL pre-training effectively shapes both the controller and Lyapunov function toward stability while also optimizing task-relevant performance objectives.

Future Work. Several directions warrant investigation: (i) *Scalability*: Extending to high-dimensional robotics problems. Unlike MILP solvers which are inherently CPU-bound, the gradient-based nature of Stage I is fully compatible with GPU parallelism, making it a promising avenue for scaling to high-dimensional robotic systems. (ii) *Richer FPL specifications*: Beyond two-term composition, FPL's hierarchical structure could encode obstacle avoidance, input constraints, or multi-modal objectives while maintaining differentiability. (iii) *Expanding the Region of Attraction*: Our method operates on a fixed domain $\mathcal{D}$ and is compatible with the ROA expansion procedure in [9], which could be applied after Stage II to maximize the certified region.

Acknowledgment

The authors thank the BU Robotics Lab for providing the computational resources and support.

References

[1] H. K. Khalil, *Nonlinear systems*. Upper Saddle River, N.J.: Prentice Hall, 2002.

[2] R. Tedrake, *Underactuated Robotics*, 2023. [Online]. Available: https://underactuated.csail.mit.edu

[3] A. Majumdar, A. A. Ahmadi, and R. Tedrake, "Control design along trajectories with sums of squares programming," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 4054–4061.

[4] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1749–1767, 2023.

[5] A. A. Ahmadi, M. Krstic, and P. A. Parrilo, "A globally asymptotically stable polynomial vector field with no polynomial lyapunov function," in *Conference on Decision and Control and European Control Conference*, 2011, pp. 7579–7580.

[6] Y.-C. Chang, N. Roohi, and S. Gao, "Neural Lyapunov Control," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[7] H. Dai, B. Landry, M. Pavone, and R. Tedrake, "Counter-example guided synthesis of neural network lyapunov functions for piecewise linear systems," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 1274–1281.

[8] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, "Lyapunov-stable neural-network control," in *Robotics: Science and Systems*, Virtual, 2021.

[9] Z. Wang, S. B. Andersson, and R. Tron, "Lyapunov neural network with region of attraction search," in *American Control Conference (ACC)*, 2024, pp. 3403–3410.

[10] C. Dawson, B. Lowenkamp, D. Goff, and C. Fan, "Learning safe, generalizable perception-based hybrid control with certificates," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1904–1911, 2022.

[11] S. Zhang, Y. Xiu, G. Qu, and C. Fan, "Compositional neural certificates for networked dynamical systems," in *Learning for Dynamics and Control Conference*. PMLR, 2023, pp. 272–285.

[12] S. Zhang and C. Fan, "Learning to stabilize high-dimensional unknown systems using lyapunov-guided exploration," in *Proceedings of the 6th Learning for Dynamics and Control Conference (L4DC)*, ser. Proceedings of Machine Learning Research, vol. 242, 2024, pp. 52–67.

[13] B. E. Mabsout, A. Abdelgawad, and R. Mancuso, "Closing the intent-to-behavior gap via fulfillment priority logic," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025, pp. 4174–4181.