A Neural-Symbolic Approach for Safe and Smooth Simplex-Based Control

Qixuan Cao
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China
51265902038@stu.ecnu.edu.cn

Peixin Wang*
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China
pxwang@sei.ecnu.edu.cn

Min Zhang
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China
zhangmin@sei.ecnu.edu.cn

***Abstract*—Ensuring learning-enabled systems to be safe remains a great challenge, especially with high-performance but black-box neural controllers in safety-critical settings. The Simplex architecture offers a pragmatic solution by switching at runtime between an unverified but performant controller and a verified but conservative one. However, the decision logic in Simplex relies on binary switching, causing abrupt action altering and consequently degrading system smoothness and performance. We propose HyPlex, a hybrid control architecture for both safe and smooth controlling. Specifically, we design a novel neural-symbolic decision logic, which hybridizes the outputs of the two types of controllers by predicting their control weights based on runtime reachability analysis results. We evaluate HyPlex on three representative benchmarks and show that, in the presence of obstacles, it improves trajectory smoothness by up to 88.27% and reduces performance degradation by up to 48.07%, compared to standard Simplex switching.**

***Index Terms*—reinforcement learning, neural-symbolic methods, safe neural control, runtime assurance**

I. Introduction

Recent advances in deep learning have led to the widespread adoption of learning-enabled systems across a growing range of domains [1], including safety-critical applications such as autonomous driving [2] and robotic manipulation [3]. However, when these systems are deployed in real-world environments, unforeseen obstacles and uncertainties may arise, potentially compromising their safety. To mitigate such risks, runtime safety assurance has emerged as a practical and effective approach for ensuring the safe operation of learning-enabled systems.

The Simplex architecture [4], as a classical example of runtime safety assurance, offers a principled approach to safeguarding learning-enabled systems. It combines a high-performance controller with a high-assurance safety controller, and employs a decision module to monitor system behavior in real time. When an impending safety violation is detected by reachability analysis [5], [6], control is switched from the high-performance controller to the safety controller to ensure safe operation. The Simplex architecture has been recently explored in increasingly complex and realistic environments for safe control, including simulated highway driving scenarios [7] and real-world robotic platforms such as the F1/10 autonomous vehicle [8]. The high-performance controllers in Simplex can be efficiently implemented as neural network (NN) controllers [9].

The main bottleneck of Simplex is that the control switch is binary (i.e., the system is governed by either the neural or the safety controller) based on pre-defined safety conditions in decision logic, leading to discontinuities or oscillations of system trajectories (see the red trajectory in Fig. 1). Such sudden switching between the two types of controllers can result in jagged and even unexpected behaviors. In addition, the switch logic of Simplex is conservative [7]. Once the safety controller takes over the system, the system reverts to the performance controller only after the safety controller has been executed for a sufficient period without detecting any unsafe behavior. As a result, these undesirable behaviors of classical Simplex can lead to performance degradation of the system.

To mitigate the abrupt hand-overs that plague classical Simplex architectures, we introduce HyPlex, a hybrid Simplex architecture that issues a single control command obtained by linearly blending the actions of a neural controller and a verified safety controller. The blending coefficient (also called *control weight*) is generated by a **neural–symbolic** module that couples forward reachability analysis with a neural network. Whereas the classical Simplex mechanism performs a purely *qualitative* switch entirely to the safety controller whenever the reachable set intersects the unsafe region within a finite horizon T—HyPlex conducts a *quantitative* analysis: it computes the earliest time instant $t \leq T$ (if it exists) at which the reachable set intersects with the unsafe region, interprets this *time-to-unsafe margin* as a safety measure, and maps the margin to a continuous control weight. [1] Because hand-crafting the margin-to-weight mapping is usually ad hoc and task-specific, we formalise the process (Section V-C) via two design principles and implement them using deep reinforcement learning to train a weighting network with reward functions that encode these principles. A generous margin allocates

*Corresponding author.

[1] If no intersection with the unsafe region is found within T, then the control authority will be fully governed by the neural controller during the next horizon T.

more authority to the learned controller, while an imminent threat shifts weight toward the safety controller, thereby preserving safety without inducing the abrupt discontinuities characteristic of binary hand-overs. The resulting hierarchy yields trajectories that are safe yet substantially smoother than those produced by the classical Simplex scheme.

Our contributions are summarized as follows:

1) We propose a novel neural-symbolic extension of Simplex for generating safe and smooth control commands to learning-enabled systems at runtime by quantitatively mixing the neural and safety controllers' outputs instead of traditional sudden switching between them.
2) We introduce two principles to characterize the relationship between the safety measure and the control weight, and present a precise and efficient method for computing hybrid control commands by training a neural network as the control weight generator based on runtime reachability analysis.
3) We demonstrate the outperformance of our approach through extensive experiments. In the case that obstacles occur, our approach improves system smoothness and meanwhile mitigates performance degradation by up to 88.27% and 48.07%, respectively.

Organization. Section III presents the problems we address, while Section IV introduces the necessary preliminaries used in this paper. Section V introduces our HyPlex architecture, and Section VI demonstrates the experimental results. Section II discusses related works, and Section VII concludes the paper and outlines the limitations and future directions.

II. Related Work

A. Neural-Symbolic Control

HyPlex aligns with the emerging paradigm of neural-symbolic integration [10], which combines the adaptability of neural networks with the rigor of symbolic systems. Cunnington et al. proposed a neural-symbolic framework for the robustness of learning from unstructured data [11]. Garnelo et al. mapped the original state to a low-dimensional symbolic representation in reinforcement learning, improving the utility of strategies [12]. Complementary to these works, our approach pursues the safety of learning-enabled systems while keeping their smoothness.

B. Online Verification

HyPlex is also a follow-up to those online verification methods for safety control of learning-enabled systems. Cofer et al. showed that learning-enabled systems' safety can be improved via runtime assurance based on the ASTM F3269-17 standard [13]. Pek et al. showed that online verification can drastically reduce traffic accidents for autonomous vehicles [14]. Brown et al. recently showed that the time overhead for quantitative reachability analysis can be reduced to 0.25 s for 100 control steps even with limited computational resources [15]. These efficient verification approaches can be integrated into our hybrid architecture.

C. Multiple Controllers Mixing

HyPlex provides an alternative strategy for mixing multiple controllers. Prior to our work, several approaches for mixing multiple controllers have been proposed. For instance, Bulut et al. proposed a fuzzy controller to blend conventional and fuzzy control strategies [16]. Kuipers and Ioannou introduced the MMAC (Multiple Model Adaptive Control) architecture, along with an adaptive mixing control supervisor that adjusts multiple controllers by assigning weights to candidate controllers [17]. Wang et al. developed COCKTAIL, a framework that trains a neural network meta-controller via reinforcement learning to integrate multiple existing control experts with dynamically assigned weights [18]. Our approach complements existing methods by introducing a principled, reachability-informed weighting mechanism that enables continuous and adaptive mixing of controllers at runtime. Ramakrishna et al. [19] proposed a weighted blending strategy to balance performance and safety, aligning with our approach. However, their method derives weights from current states only, whereas our work integrates reachability analysis to incorporate future state projections, thereby prioritizing safety over performance during critical transitions. Moreover, their method restricts the weights to discrete values, while ours allows a fully continuous range, thereby enabling richer blending possibilities and smoother transitions between controllers.

III. Problem Statement

Although the Simplex architecture safeguards systems by toggling between a high-performance controller and a safety controller, its *binary switching mechanism* often induces abrupt control transitions. These sudden hand-overs can introduce signal discontinuities, trigger oscillations near the switching threshold, and degrade performance when the safety controller is invoked too conservatively. Hence, in this work, we aim to solve the problem below.

Problem. Develop a new decision module, in real time, to determine how much authority to allocate to each controller so that safety remains intact, task performance is maximized whenever conditions permit, and the system trajectory evolves smoothly.

As shown in Section V, we address this problem with a **neural–symbolic approach** that couples formal reachability analysis with a neural network, enabling principled and adaptive blending of the controllers without sacrificing safety.

Illustrative Example. Fig. 1 illustrates a driving scenario comparing the Simplex architecture [20] and our HyPlex method. As shown by the red trajectory, when an obstacle appears, Simplex abruptly switches control to the safety controller, causing a sudden right turn. Once the obstacle is cleared, control switches back, resulting in another sudden turn. In contrast, HyPlex behaves differently: as the vehicle approaches the obstacle, it progressively shifts control authority from the learned controller to the safety controller, then gradually restores that authority after the obstacle is passed. This results in a smoother avoidance maneuver, as depicted by the blue trajectory.

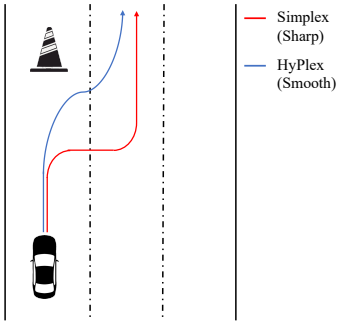


Fig. 1: A driving case

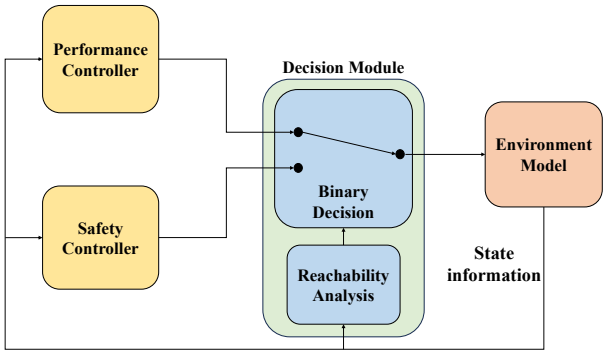


Fig. 2: Simplex Architecture

IV. Preliminaries

In this section, we present the control system used in this work and introduce the Simplex architecture. We denote by $\mathbb{N}$ and $\mathbb{R}$ the sets of all natural numbers excluding 0 and real numbers, respectively.

A. Control Systems

A control system governs the behavior of dynamic entities through mathematically defined interactions among states, actions, and the environment. Formally, we model such a system as a tuple $C = (S, S_0, S_g, S_u, A, \pi, f)$, where:

(i) $S \subseteq \mathbb{R}^n$ represents the (potentially infinite and continuous) state space,
(ii) $S_0 \subseteq S$ is the set of initial states,
(iii) $S_g \subseteq S$ is the set of target states,
(iv) $S_u \subseteq S$ is the set of unsafe states, satisfying $S_u \cap S_g = \emptyset$,
(v) A denotes the set of actions,
(vi) $\pi : S \to A$ is the control policy that maps states to actions,
(vii) $f : S \times A \to \dot{S}$ is the system dynamics.

The system evolves in discrete time steps $t \in \mathbb{N}_0$. Each step spans a fixed duration δ, during which the selected action remains constant. At each step, the system observes the current state $s_t \in S$ and gets an action $a_t = \pi(s_t)$ from the control policy π. The state evolves according to the system dynamics $\dot{s}(\tau) = f(s(\tau), a_t)$ for $\tau \in [0, \delta]$, and the system transitions to the next state $s_{t+1} = s_t + \int_0^\delta f(s(\tau), a_t)\, d\tau$.

Trajectories and Episodes. The system begins its interaction with the environment from an initial state $s_0 \in S_0$. As it acts and observes over time, it produces a sequence of states—referred to as a *trajectory*—denoted by $\xi = \{s_t\}_{t \in \mathbb{N}_0}$, which captures the system's behavioral evolution under the control policy and the environment's dynamics. In deep reinforcement learning [21], a reward function $R : S \times A \to \mathbb{R}$ provides feedback for learning. At each time step, the system receives a scalar reward $r_{t+1} = R(s_t, a_t)$. An *episode* is defined as a sequence of state-action-reward transitions, typically represented as $\varepsilon = \{(s_t, a_t, r_{t+1})\}_{t \in \mathbb{N}_0}$.

B. Simplex Architecture

The Simplex architecture [4] is a control framework designed for runtime safety assurance. Concretely, it consists of three principal components: Performance Controller (PC), Safety Controller (SC) and Decision Module (DM), as shown in Fig. 2. In this work, we consider a deep reinforcement learning (DRL)-based policy π_{RL} as the PC, and a rule-guided safety policy π_{safe} as the SC. The decision module of Simplex selects the control command as follows:

$$a = \lambda_B \cdot a_{RL} + (1 - \lambda_B) \cdot a_{safe}, \tag{1}$$

where $a_{RL} = \pi_{RL}(s)$ and $a_{safe} = \pi_{safe}(s)$ are the actions given by the neural and safety controllers, respectively, for any state $s \in S$, and the control weight $\lambda_B \in \{0, 1\}$ is a Boolean flag indicating which controller is used. When control commands are m-dimensional, λ_B is also an m-dimensional vector where each element is either 1 or 0.

V. Neural-Symbolic Approach

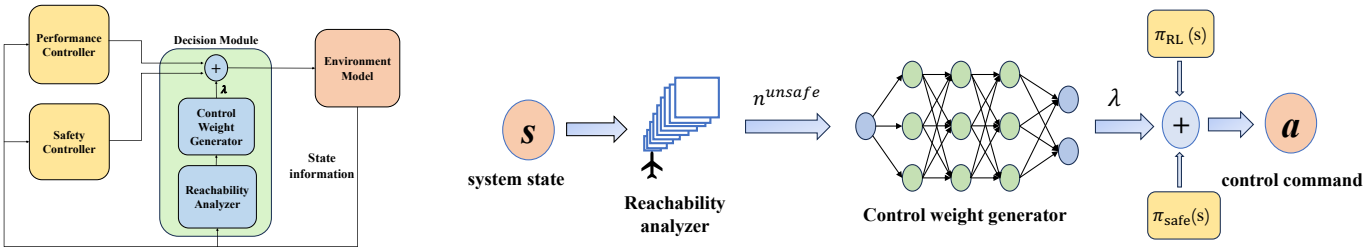
A. HyPlex Architecture

As mentioned in Section III, the switch logic of the decision module in Simplex is binary, causing abrupt action altering and consequently degrading system smoothness and performance, due to the binary control weight $\lambda_B \in \{0, 1\}$ (see Eq. (1)). To ease the presentation, here we consider the one-dimensional case, but it can be straightforwardly extended to high-dimensional cases.

Key Idea of HyPlex. To smooth the switching between the performance controller and the safety controller, we propose our HyPlex architecture. The key idea here is that we linearly blend the outputs of the two controllers, i.e., the binary control weight λ_B in Eq. (1) is replaced by a real-valued continuous control weight $\lambda_C \in [0, 1]$ so that a hybrid control command is obtained as $a = \lambda_C \cdot a_{RL} + (1 - \lambda_C) \cdot a_{safe}$, as depicted in Fig. 3a.

Smoothness via continuous control weights. The smoothness of HyPlex is based on two aspects via the continuous control weight λ_C.

- **Relieve abrupt switch.** When λ_B suddenly transfers from 1 to 0 (or 0 to 1) in Simplex, the gradual change of λ_C in HyPlex relieves the magnitude of action altering, which possibly produces a smoother control command (see Proposition 1).



(a) HyPlex Architecture (b) Decision Module

Fig. 3: HyPlex architecture and details of the neural-symbolic decision module.

- **Enlarge candidate action sets.** When $\lambda_B = 1$ or 0 in Simplex, i.e., the action to be taken is either from $\{a_{RL}\}$ or $\{a_{safe}\}$, the continuous λ_C in HyPlex implies a candidate action set with more than one element, which makes it possible to get a smoother control command (see Proposition 2).

Proposition 1. *The magnitude of action altering in HyPlex is no more than that in Simplex when the binary switching happens.*

Proof. When the control authority is switched from PC to SC at time t in Simplex, i.e., λ_B transfers from 1 to 0, we have the magnitude of action change in Simplex as $\Delta = |\pi_{RL}(s_t) - \pi_{safe}(s_t)|$. As the continuous λ_C in HyPlex decreases from 1 to some constant $\lambda \in [0, 1)$, the magnitude of action change in HyPlex is:

$$\begin{aligned}\Delta' &= |\pi_{RL}(s_t) - [\lambda \cdot \pi_{RL}(s_t) + (1-\lambda) \cdot \pi_{safe}(s_t)]| \\ &= (1-\lambda) \cdot |(\pi_{RL}(s_t) - \pi_{safe}(s_t))| \\ &= (1-\lambda) \cdot \Delta.\end{aligned}$$

As $\lambda \in [0, 1)$, we have that $\Delta \geq \Delta'$. The same claim holds when we consider the switch from SC to PC in Simplex. □

Proposition 2. *HyPlex enables a larger candidate action set than Simplex with respect to any specific state.*

Proof. Given a state $s \in S$, in Simplex, when $\lambda_B = 1$, the action to be taken is from the singleton set $\{a = a_{RL}\}$. In contrast, HyPlex allows actions to be any convex combination of a_{RL} and a_{safe}, which enlarges the candidate action set as $\{a = \lambda_C \cdot a_{RL} + (1-\lambda_C) \cdot a_{safe} \mid \lambda_C \in [0, 1]\}$. The same claim holds when $\lambda_B = 0$. □

Neural-symbolic Decision Module. To generate the hybrid control command, we propose a neural-symbolic decision module, as illustrated in Fig. 3b. The decision module comprises two components:

- **Symbolic module - Reachability analyzer**: computes a quantitative safety measure n^{unsafe} (Section V-B),
- **Neural module - Control weight generator**: predicts the continuous control weight λ_C via a neural network π_{weight} (Section V-C).

Algorithm 1: Neural-symbolic Decision Module

Input: Current system state s, Maximal time step N
Output: Control command a

1: $a_{RL} \leftarrow \pi_{RL}(s)$; $a_{safe} \leftarrow \pi_{safe}(s)$
2: $n^{unsafe} \leftarrow TimetoUnsafe(s, N)$
3: $\lambda \leftarrow \begin{cases} \pi_{\text{weight}}(n^{unsafe}), & \text{if } n^{unsafe} \neq +\infty, \\ 1, & \text{otherwise.} \end{cases}$
4: $a \leftarrow \lambda \cdot a_{RL} + (1-\lambda) \cdot a_{safe}$
5: **return** a

Algorithm 1 describes the decision module. Given the current system state s, we calculate the actions of the performance and safety controllers, a_{RL} and a_{safe} (Line 1). Line 2 applies the symbolic module and computes n^{unsafe} using the function $TimetoUnsafe(s, N)$, where N is a hyperparameter deciding the maximal time step of reachability analysis. Line 3 calculates the control weight: if $n^{unsafe} \neq +\infty$, λ is generated by $\pi_{\text{weight}}(n^{unsafe})$; otherwise, we set $\lambda = 1$. The final hybrid control command is given by $a = \lambda \cdot a_{RL} + (1-\lambda) \cdot a_{safe}$ (Line 4).

B. Reachability Analysis

Given a dynamic system, reachability analysis [6], [22] aims to compute the set of system states reachable from a set of initial states. In this work, considering a control system $C = (S, S_0, S_g, S_u, A, \pi, f)$ with dynamics $\dot{s} = f(s, \pi(s))$, starting from an initial state s_0, the state at time $t = n\delta + t'$ for some integer $n \geq 0$ and $0 \leq t' \leq \delta$ is defined as:

$$\varphi_f(s_0, \pi, t) = s_n + \int_0^{t'} f(s(\tau), \pi(s_n))\, d\tau, \tag{2}$$

where $s_{i+1} = s_i + \int_0^{\delta} f(s(\tau), \pi(s_i))\, d\tau$ for all $i \in \{0, \ldots, n-1\}$. The reachable set of states w.r.t.s_0 from the *i-th* time step to the *j-th* time step is denoted as:

$$Reach(s_0, i, j) = \{\varphi_f(s_0, \pi, t) \mid t \in [i\delta, j\delta], i \leq j\}. \tag{3}$$

Specifically, the reachable set of states w.r.t.s_0 during the *n-th* time step and over all the n time steps can be expressed as follows:

$$Reach(s_0, n-1, n) = \{\varphi_f(s_0, \pi, t) \mid t \in [(n-1)\delta, n\delta]\} \tag{4}$$

Algorithm 2: Safety Measure $TimetoUnsafe$

Input: Current system state s, Maximal time step N
Output: Time-to-unsafe margin n^{unsafe}

Initialize $n^{unsafe} \leftarrow +\infty$
for $n \leftarrow 1$ **to** N **do**
  $S_{reach} \leftarrow Reach(s, n-1, n)$
  if $S_{reach} \cap S_u \neq \emptyset$ **then**
    $n^{unsafe} \leftarrow n$
    break
  end if
end for
return n^{unsafe}

$$Reach(s_0, 0, n) = \{\varphi_f(s_0, \pi, t) \mid t \in [0, n\delta]\} \tag{5}$$

In Simplex, the control weight λ_B in Eq. (1) can be determined based on reachability analysis. At each time step, the decision module performs reachability analysis over the next N time steps using the action proposed by π_{RL}, which is $Reach(s, 0, N)$ as defined in Eq. (5). Given the set of unsafe states S_u, the control weight λ_B is defined qualitatively as:

$$\lambda_B = \begin{cases} 1, & \text{if } Reach(s, 0, N) \cap S_u = \emptyset, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

In HyPlex, to define the continuous control weight λ_C, we need a quantitative analysis of reachability computation results, instead of the qualitative one. HyPlex considers the first time step $n^{unsafe} \in [1, N]$ at which the system, when controlled by the performance controller, violates safety constraints (i.e., $Reach(s, 0, n^{unsafe}) \cap S_u \neq \emptyset$). This time-to-unsafe margin can be interpreted as a safety measure. Formally, n^{unsafe} for a state s and a time step N is defined as follows:

$$\begin{aligned} n^{unsafe} = \min\{n \in \{1, 2, \ldots, N\} \mid \\ Reach(s, n-1, n) \cap S_u \neq \emptyset\}, \end{aligned} \tag{7}$$

where $Reach(s, n-1, n)$ is the reachable set during the *n-th* time step defined in Eq. (4) and S_u is the set of unsafe states. If every intersection is empty, then n^{unsafe} keeps the default value $+\infty$, indicating that the system is sufficiently safe.

The process of calculating n^{unsafe} is denoted by the function $TimetoUnsafe(s, N)$, which is depicted in Algorithm 2. In the process of calculating the reachable set over all N time steps, which is $Reach(s, 0, N)$, HyPlex extracts the current reachable set $Reach(s, n-1, n)$ for each time step, and checks whether $Reach(s, n-1, n)$ has any intersection with S_u. If the intersection exists, n^{unsafe} is then determined as the current n and we exit the current loop, as there is no need to compute the subsequent reachable sets.

C. Control Weight Generator

In this section, we introduce the neural part of our neural-symbolic decision module. As shown in Fig. 3b, the control weight generator π_{weight} is a neural network that takes the time-to-unsafe margin n^{unsafe} as input and outputs the continuous control weight λ. Note that here we omit the subscript $''C''$ of the control weight because the context is clear.

Fix the PC π_{RL} and the SC π_{safe}. The training of π_{weight} is conducted using the reinforcement learning framework. At each time step t, the system observes its current state $s_t \in S$. Reachability analysis is then performed based on s_t to compute $n_t^{unsafe} = TimetoUnsafe(s_t, N)$. Subsequently, n_t^{unsafe} is fed into π_{weight} to generate the continuous control weight λ_t. The hybrid action is computed as $a_t = \lambda_t \cdot \pi_{RL}(s_t) + (1 - \lambda_t) \cdot \pi_{safe}(s_t)$. Applying a_t to the system dynamics, the system transitions to the next state s_{t+1} and receives a scalar reward $\tilde{r}_t$ derived from the newly designed reward function $\tilde{R}$ for training π_{weight}. Based on s_{t+1}, a new time-to-unsafe margin is calculated by $n_{t+1}^{unsafe} = TimetoUnsafe(s_{t+1}, N)$. In this way, an episode $\{(n_t^{unsafe}, \lambda_t, n_{t+1}^{unsafe}, \tilde{r}_t)\}_{t \in \mathbb{N}_0}$ of these transition tuples is collected.

The core of conducting this reinforcement learning process lies in designing the reward function. For a deterministic system, the reward function R in reinforcement learning takes the form of $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In the training of the control weight generator, the new reward function becomes $\tilde{R} : \mathbb{N} \times [0, 1] \to \mathbb{R}$, e.g., $\tilde{r}_t = \tilde{R}(n_t^{unsafe}, \lambda_t)$. To construct this reward function, we introduce two principles here to characterize the relationship between n^{unsafe} and λ.

Reward Design Principles. The relationship between n^{unsafe} and λ is characterized by the following two principles:

- **Maximize performance while ensuring safety**: λ should be as large as possible as long as the system will not enter unsafe areas.
- **Increase conservativeness as safety margin decreases**: The closer the predicted reachable set is to unsafe regions (indicated by a smaller n^{unsafe}), the smaller λ should be, thereby increasing the influence of the safety controller.

Intuition. [23, Theorem 1] shows that the upper bound of performance degradation will be lower as long as the policy is changed less. To preserve the performance benefits of the learning-enabled controller, the system should favor high values of λ (i.e., $1-\lambda$ is small) as long as the safety constraints are obeyed (i.e., n^{unsafe} is sufficiently large), which is claimed in the first principle. In contrast, the second principle focuses on the relative adjustment of λ in response to varying levels of risk: as n^{unsafe} decreases, λ should be progressively reduced to increase the conservativeness of the control policy.

Following these two principles, we design the corresponding reward function below:

$$\tilde{r} = \begin{cases} p - \log(n^{unsafe}) \cdot \mathbf{1}^\top(\mathbf{1} - \boldsymbol{\lambda}), & \text{if } n^{unsafe} \neq +\infty, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

$$p = \begin{cases} -K, & \text{if } s \in S_u, \\ 0, & \text{otherwise,} \end{cases} \tag{9}$$

Algorithm 3: Control Weight Generator Training

Input: Initial control weight generator π_{weight_0}
Output: Control weight generator π_{weight_M}

Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$
for iteration $i \leftarrow 0$ **to** $M-1$ **do**
  Initialize environment state s_0
  Calculate n_0^{unsafe}, λ_0 and a_0 for s_0 based on Algorithm 1
  for each environment step t, until done **do**
    $s_{t+1}, \tilde{r}_t \leftarrow env(a_t)$
    Calculate n_{t+1}^{unsafe}, λ_{t+1} and a_{t+1} for s_{t+1} (Algorithm 1)
    Collect transition $(n_t^{unsafe}, \lambda_t, n_{t+1}^{unsafe}, \tilde{r}_t)$
    Add $(n_t^{unsafe}, \lambda_t, n_{t+1}^{unsafe}, \tilde{r}_t)$ to $\mathcal{B}$
  end for
  Sample a batch of transitions $\mathcal{D}$ from $\mathcal{B}$
  Update π_{weight_i} based on $\mathcal{D}$ and get $\pi_{\text{weight}_{i+1}}$
end for
return π_{weight_M}

where p denotes the safety penalty. If the system enters the unsafe set, a large negative constant penalty $-K$ is applied; otherwise, no penalty is imposed (i.e., $p = 0$).

The safety penalty p keeps the system away from safety violations. Then another penalty term $1-\lambda$ corresponds to the first principle, to minimize the magnitude of intervention from the safety controller. The coefficient of this term $\log(n^{unsafe})$ corresponds to the second principle. A logarithmic scale is used to avoid the overly strong effect of linear weights. The overall training algorithm is shown in Algorithm 3, where M is the maximal iteration and env is the environment. The training process follows standard reinforcement learning but with a symbolically represented environment interface and a redesigned reward function.

VI. Experiments

A. Experimental settings

HyPlex is designed as an extension to the Simplex architecture, serving as an add-on module that can enhance a broad class of Simplex-based runtime assurance methods. Accordingly, our experimental evaluation focuses on a state-of-the-art Simplex implementation [20] as the baseline. By integrating HyPlex on top of the same Simplex framework, we aim to isolate and quantify the performance improvements brought by HyPlex, while keeping all other components unchanged.

In our implementation of the HyPlex architecture, we use DDPG [24] as the backend DRL algorithm for training the performance controller π_{RL} and the control weight generator π_{weight}, and choose POLAR-Express [6] as the reachability analysis tool in the decision module. The safety controller π_{safe} is constructed by handcrafted rule-based logic, which is specifically tailored to steer the agent away from unsafe sets in the state space. We implement both π_{RL} and π_{weight} using multi-layer perceptrons (MLPs) with two hidden layers, each containing 64 neurons.

Benchmarks. We choose three classical benchmarks, including B2 from [22], Mountain-Car-Continuous (MCC) from OpenAI Gym [25] and Spaceship-Corridor (SSC) from [26]. The state space and the action space of these benchmarks are continuous. B2 and MCC have a one-dimensional action space, while SSC has a two-dimensional action space. For B2, MCC and SSC, the time step length is set to 0.05, 0.1 and 0.2 seconds, respectively.

We first train π_{RL} on the original environment without unsafe sets, i.e., the task of π_{RL} is to reach the goal regions. Then we add unsafe sets to the environment, and the new tasks of Simplex and HyPlex are to guide the agent from the initial set to the goal region and avoid unsafe sets during execution.

The Simplex policy follows the implementation in [20], which is:

$$\pi_{\text{Simplex}}(s) = \begin{cases} \pi_{RL}(s), & \text{if } Reach(s, 0, N) \cap S_u = \emptyset, \\ \pi_{safe}(s), & \text{otherwise,} \end{cases} \tag{10}$$

where N is the maximal time step used in reachability analysis (see Section V-B). And our HyPlex policy is:

$$\begin{aligned} \pi_{\text{HyPlex}}(s) = \pi_{\text{weight}}(n^{unsafe}) \cdot \pi_{RL}(s) \\ +(1 - \pi_{\text{weight}}(n^{unsafe})) \cdot \pi_{safe}(s). \end{aligned} \tag{11}$$

Platform. All experiments were conducted on a computer equipped with an Intel Core i9-13900K CPU and 64GB RAM, running Ubuntu 20.04.6 LTS.

B. Evaluation

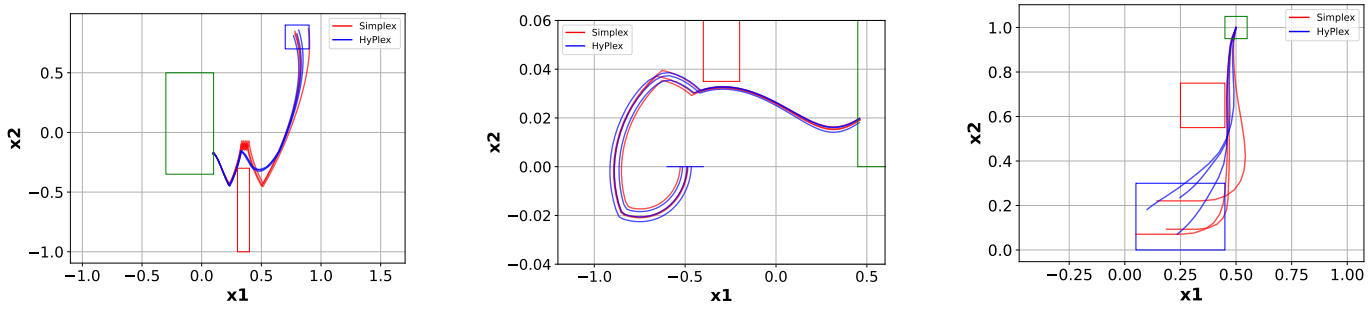
We compare HyPlex and Simplex under identical implementations of π_{RL}, π_{safe}, and the reachability analysis tool. Their performance is evaluated using the following metrics:

- *Reward*: the average cumulative rewards of all episodes.
- *Smoothness*: a smoothness metric is defined as $\mathbf{S} = \mathbf{S}_1 \times \mathbf{S}_2$, where $\mathbf{S}_1$ and $\mathbf{S}_2$ [27], [28] are two metrics to evaluate the smoothness of actions, i.e.,

$$\mathbf{S}_1 = \frac{1}{n-1} \left(\sum_{i=1}^{n-1} (a_{i+1} - a_i)^2 \right), \tag{12}$$

$$\mathbf{S}_2 = 100 \times \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (a_i - \bar{a})^2}, \tag{13}$$

  where n denotes the total number of steps in an episode, a_i is the action taken at step i, and $\bar{a}$ is the average action value over the entire episode. Intuitively, $\mathbf{S}_1$ measures the temporal smoothness of the actions, and $\mathbf{S}_2$ denotes the standard deviation of actions (scaled by a factor of 100 to improve numerical resolution). We calculate the average $\mathbf{S}$ over all episodes. Note that the smaller $\mathbf{S}$ is, the smoother the system trajectory is.
- *Violation*: the number of safety violations by simulation in all episodes.



(a) B2 (Opt) (b) MCC (Opt) (c) SSC (Opt)

Fig. 4: Trajectories of states. (HyPlex vs. Simplex-Opt. The blue boxes are the initial sets, the green boxes are the goal regions and the red boxes are the unsafe sets.)

TABLE I: Performance of the RL Policy in original environments without unsafe sets.

Tasks	Reward	Smoothness	Violation
B2	63.370 ± 2.429	5.982 ± 1.212	0
MCC	93.494 ± 0.029	2.651 ± 0.400	0
SSC	1000.0 ± 0.0	5.332 ± 2.641 9.641 ± 1.619	0

TABLE II: Performance Evaluation (**N**: the time horizon in reachability analysis).

Tasks		N	Reward	Smoothness	Violation
B2	Original	/	-32.02 ± 2.50	0.18 ± 0.52	1000
	Simplex-Match	10	-102.11 ± 1.42	271.66 ± 36.90	0
	Simplex-Opt	4	58.96 ± 2.49	55.75 ± 11.82	0
	HyPlex	10	$\mathbf{61.08 \pm 2.27}$	$\mathbf{6.54 \pm 0.41}$	0
MCC	Original	/	-4.44 ± 0.25	4.06 ± 0.37	1000
	Simplex-Match	10	91.44 ± 0.41	14.80 ± 3.97	0
	Simplex-Opt	6	91.66 ± 0.37	12.45 ± 2.90	0
	HyPlex	10	$\mathbf{92.09 \pm 0.39}$	$\mathbf{8.28 \pm 2.90}$	0
SSC	Original	/	117.0 ± 321.42	11.00 ± 6.62 6.51 ± 4.53	883
	Simplex-Match	10	1000.0 ± 0.0	17.67 ± 5.25 12.10 ± 5.02	0
	Simplex-Opt	10	1000.0 ± 0.0	17.67 ± 5.25 12.10 ± 5.02	0
	HyPlex	10	$\mathbf{1000.0 \pm 0.0}$	$\mathbf{8.53 \pm 3.22}$ $\mathbf{9.19 \pm 2.20}$	0

We evaluate each setting over 1000 episodes. The performance of the RL policy in original environments without unsafe sets serves as a reference, as shown in Table I. For Simplex, we report two variants. The first is Simplex-Match, which uses the same 10-step horizon as HyPlex. The second is Simplex-Opt, for which we search over horizons from 1 to 10 steps and select the one with the highest $Reward$ and smallest **S**. The optimal horizon for Simplex-Opt is 4 in B2, 6 in MCC, and 10 in SSC. We compare four policies: the original RL policy π_{RL} (Original), Simplex-Match, Simplex-Opt, and our proposed HyPlex policy in the environments with unsafe sets, as shown in Table II.

Safety. In contrast to the original RL policy, which leads to a large number of safety violations, Simplex-Match, Simplex-Opt and HyPlex all achieve zero safety violations in all three tasks.

Performance. HyPlex achieves the best performance in terms of reward. We use the performance in the original environments without unsafe sets as a reference. Compared to Simplex-Opt, HyPlex reduces the performance degradation caused by obstacle avoidance by 48.07% in B2 and 23.30% in MCC. In particular, Simplex-Match in the B2 environment is too conservative and fails to reach the goal region within the maximum step limit, resulting in poor rewards.

Smoothness. HyPlex achieves significant improvements in smoothness. Compared with Simplex-Opt, it achieves a 33.49% improvement in MCC, and improvements of 51.74% and 24.08% in the two action dimensions of SSC, respectively. The maximum improvement occurs in B2, which is 88.27%, as shown in Table II. Fig. 4 illustrates the system trajectories in the three benchmarks, with the results of Simplex-Opt and HyPlex depicted in red and blue, respectively. We can see that HyPlex produces smoother trajectories. We highlight the case in B2. As shown by the red trajectories in Fig. 4(a), Simplex exhibits oscillatory behavior, where the agent hovers near the decision boundary, triggering frequent controller switches. In contrast, HyPlex, with its hybrid control strategy, effectively mitigates oscillation, as reflected in the smoother blue trajectories.

Hyperparameter Selection. The maximal time horizon N used in reachability analysis is a hyperparameter that affects both the behavior and computational cost of HyPlex. A larger N results in higher time consumption due to the expanded analysis scope. We empirically investigate how to select an appropriate value of N. As shown in Table III, we evaluate HyPlex under different settings of N. When increasing N from 5 to 10, we observe notable improvements in performance, smoothness, and safety. A longer time horizon for reachability analysis enables HyPlex to explore more effective strategies. However, increasing N further to 15 yields diminishing returns, with results comparable to those at $N = 10$. Considering both performance and runtime overhead, we select $N = 10$ as a balanced and effective choice.

TABLE III: HyPlex with different reachability horizons (N: the time horizon).

Tasks	N	Reward	Smoothness	Violation
B2	5	56.30 ± 2.65	41.03 ± 10.46	0
	10	$\mathbf{61.08 \pm 2.27}$	6.54 ± 0.41	0
	15	60.37 ± 2.29	$\mathbf{6.35 \pm 0.28}$	0
MCC	5	78.42 ± 33.30	12.47 ± 2.72	138
	10	92.09 ± 0.39	8.28 ± 2.90	0
	15	$\mathbf{92.20 \pm 0.40}$	$\mathbf{8.03 \pm 2.88}$	0
SSC	5	1000.0 ± 0.0	26.16 ± 10.11 9.36 ± 4.26	0
	10	$\mathbf{1000.0 \pm 0.0}$	8.53 ± 3.22 $\mathbf{9.19 \pm 2.20}$	0
	15	1000.0 ± 0.0	$\mathbf{8.26 \pm 2.91}$ 9.26 ± 2.19	0

VII. Conclusion

In this paper, we proposed HyPlex, a neural-symbolic approach for Simplex-based control that balances safety and smoothness by replacing abrupt switching between controllers with a smooth hybrid mechanism. Experiments showed that HyPlex improves smoothness and mitigates performance degradation, while maintaining zero safety violations across all evaluated benchmarks. HyPlex showcases a quantitative integration of neural computation and symbolic reasoning for balancing high-performance and high-assurance controllers.

As for future work, a forthcoming step is to extend our approach to richer quantitative safety analysis metrics, supporting more complex temporal safety requirements. We also intend to extend our approach by developing runtime safety verification algorithms that provide theoretical safety guarantees for the integrated strategy.

References

[1] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: an overview," in *Proceedings of SAI intelligent systems conference*. Springer, 2016, pp. 426–440.

[2] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE transactions on intelligent transportation systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[4] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The simplex architecture for safe online control system upgrades," in *Proceedings of the 1998 American Control Conference (ACC)*, vol. 6. IEEE, 1998, pp. 3504–3508.

[5] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*. Springer, 2013, pp. 258–263.

[6] Y. Wang, W. Zhou, J. Fan, Z. Wang, J. Li, X. Chen, C. Huang, W. Li, and Q. Zhu, "Polar-express: Efficient and precise formal reachability analysis of neural-network controlled systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 3, pp. 994–1007, 2023.

[7] S. Chen, Y. Sun, D. Li, Q. Wang, Q. Hao, and J. Sifakis, "Runtime safety assurance for learning-enabled control of autonomous driving vehicles," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8978–8984.

[8] P. Musau, N. Hamilton, D. M. Lopez, P. Robinette, and T. T. Johnson, "On using real-time reachability for the safety assurance of machine learning controllers," in *IEEE International Conference on Assured Autonomy (ICAA)*. IEEE, 2022, pp. 1–10.

[9] D. T. Phan, R. Grosu, N. Jansen, N. Paoletti, S. A. Smolka, and S. D. Stoller, "Neural simplex architecture," in *International Symposium on NASA Formal Methods (NFM)*. Springer, 2020, pp. 97–114.

[10] T. R. Besold, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K.-U. Kühnberger, L. C. Lamb, P. M. V. Lima, L. de Penning, G. Pinkas *et al.*, "Neural-symbolic learning and reasoning: A survey and interpretation 1," in *Neuro-symbolic artificial intelligence: The state of the art*. IOS press, 2021, pp. 1–51.

[11] D. Cunnington, M. Law, J. Lobo, and A. Russo, "Ffnsl: feed-forward neural-symbolic learner," *Machine Learning*, vol. 112, no. 2, pp. 515–569, 2023.

[12] M. Garnelo, K. Arulkumaran, and M. Shanahan, "Towards deep symbolic reinforcement learning," *arXiv preprint arXiv:1609.05518*, 2016.

[13] D. Cofer, I. Amundson, R. Sattigeri, A. Passi, C. Boggs, E. Smith, L. Gilham, T. Byun, and S. Rayadurgam, "Run-time assurance for learning-enabled systems," in *12th International Symposium on NASA Formal Methods (NFM)*. Springer, 2020, pp. 361–368.

[14] C. Pek, S. Manzinger, M. Koschi, and M. Althoff, "Using online verification to prevent autonomous vehicles from causing accidents," *Nature Machine Intelligence*, vol. 2, no. 9, pp. 518–528, 2020.

[15] R. Brown, L. V. Nguyen, W. Xiang, M. Wolf, and H.-D. Tran, "Perception-based quantitative runtime verification for learning-enabled cyber-physical systems," in *Proceedings of the ACM/IEEE 16th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2025)*, 2025, pp. 1–11.

[16] C. Bulut, B. Eksin, M. Güzelkaya, T. Kumbasar, and E. Yeşil, "Fuzzy blending hybrid structure with fuzzy and conventional pid controllers," in *Proceedings of the International Conference on Modeling and Applied Simulation, MAS*, 2009.

[17] M. Kuipers and P. Ioannou, "Multiple model adaptive control with mixing," *IEEE transactions on automatic control*, vol. 55, no. 8, pp. 1822–1836, 2010.

[18] Y. Wang, C. Huang, Z. Wang, S. Xu, Z. Wang, and Q. Zhu, "Cocktail: Learn a better neural network controller from multiple experts via adaptive mixing and robust distillation," in *58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 397–402.

[19] S. Ramakrishna, C. Harstell, M. P. Burruss, G. Karsai, and A. Dubey, "Dynamic-weighted simplex strategy for learning enabled cyber physical systems," *Journal of systems architecture*, vol. 111, p. 101760, 2020.

[20] F. Yang, S. S. Zhan, Y. Wang, C. Huang, and Q. Zhu, "Case study: Runtime safety verification of neural network controlled system," in *International Conference on Runtime Verification*. Springer, 2024, pp. 205–217.

[21] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[22] R. Ivanov, T. Carpenter, J. Weimer, R. Alur, G. Pappas, and I. Lee, "Verisig 2.0: Verification of neural network controllers using taylor model preconditioning," in *International Conference on Computer Aided Verification (CAV)*. Springer, 2021, pp. 249–262.

[23] W. Zhou, R. Gao, B. Kim, E. Kang, and W. Li, "Runtime-safety-guided policy repair," in *20th International Conference on Runtime Verification (RV)*. Springer, 2020, pp. 131–150.

[24] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[25] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv preprint arXiv:1606.01540*, 2016.

[26] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018.

[27] A. Raffin, J. Kober, and F. Stulp, "Smooth exploration for robotic reinforcement learning," in *Conference on robot learning*. PMLR, 2022, pp. 1634–1644.

[28] J. Chen, C. Yu, Y. Xie, F. Gao, Y. Chen, S. Yu, W. Tang, S. Ji, M. Mu, Y. Wu *et al.*, "What matters in learning a zero-shot sim-to-real rl policy for quadrotor control? a comprehensive study," *IEEE Robotics and Automation Letters*, 2025.