

# Towards Verification-Driven Control Learning

Stefan Mitsch

School of Computing, DePaul University, Chicago, Illinois 60604–2301

Email: smitsch@depaul.edu

**Abstract**—The increasing use of machine learning in systems development means that guarantees about the behavior of learned components must be a vital part of the development process. In autonomous systems, these types of guarantees are not properties of a neural network in isolation but require analyzing its interaction with the operating environment. Open-loop verification addresses neural networks in isolation and is thus incapable of providing the required guarantees; closed-loop verification tools offer limited generality (e.g., finite time analysis) and include extensive correctness-critical code. Moreover, separating verification activities from development activities makes cross-fertilization challenging. This paper sketches a framework to address these gaps with a logic-based approach for closed-loop theorem proving that references abstractions built from open-loop analysis of neural networks. The framework combines theorem proving with runtime monitoring, neural network verification, and reward shaping for reinforcement learning.

## I. INTRODUCTION

The increasing use of machine learning in control systems development—particularly in safety-critical domains such as avionics, ground transportation, and industrial automation—means that guarantees about the behavior of learned components must be a vital part of the development process (e.g., safe motion of a robotic arm, controlled by a neural network). Such guarantees are not properties of a neural network in isolation but require analyzing its interaction with other components and the operating environment.

Current verification processes, however, have significant limitations. Open-loop verification scales to neural networks of considerable size, but addresses neural networks in isolation and is thus incapable of providing the required guarantees. Closed-loop verification approaches based on model checking, reachability analysis, and simulation, in contrast, analyze neural networks and their interaction with the environment, but the large amount of correctness-critical code in tool implementations and their restriction to finite time limits the trustworthiness and generality of the analysis results. Moreover, today’s open- and closed-loop approaches separate verification activities from development activities, which makes cross-fertilization challenging. We sketch a logic-based approach to address these gaps with hybrid systems models that describe neural networks abstractly through control envelopes.

## II. BACKGROUND AND RELATED WORK

Safe reinforcement learning (see [1, 2] for a survey) can be approached from several perspectives. We focus on formal methods to provide guarantees about control neural networks. Existing shielding methods for control learning (e.g., [3–8]) provide formal guarantees by constraining learning to known

correct actions, which limits optimization to the confined space of engineered and verified options. Closed-loop neural network verification bases on reachability analysis for neural network controllers composed with continuous plant models (e.g., [4, 9]), but their restriction to finite time<sup>1</sup> limits the generality of the obtained results and requires trusting a large soundness-critical codebase. Open-loop neural network verification (e.g., [10–14]) scales well but ignores the interaction with an environment. Separating the concerns between infinite-time system safety analysis and neural network verification [15] is a major step in obtaining practical neural network verification tools for autonomous systems.

In the following sections, we elaborate on integrating existing formal methods and tool implementations, and sketch future challenges. We base our discussion on differential game logic (dGL [16]) for its expressiveness and availability of practical verification tools. Logic-based formal methods, today, present fragmented solutions to the challenges of provably correct reinforcement learning: (i) theorem proving [17] provides strong correctness guarantees for formal specifications of control actions and their physical effects; (ii) control envelope synthesis [18] identifies conditions under which actions are guaranteed to maintain safety; (iii) component-based techniques [19] and communication-based modeling [20] express interfaces and contracts between the components of a system; (iv) verified runtime monitoring and model validation [21] filters unsafe control decisions and monitors the environment for satisfying assumptions; (v) neural network verification [15] analyzes neural networks on all possible inputs for their infinite-time compliance with a nonlinear specification; (vi) explainable AI techniques [22] summarize the input-output behavior of neural networks to extract control envelopes; and (vii) reward shaping [23] generates parts of a reward function from formal specifications.

## III. THEOREM PROVING & NEURAL NETWORK ANALYSIS

The neural network analysis process in Fig. 1 starts from a partial formal specification, which may list control scenarios but lack the corresponding action to execute, or list actions without knowing when they are safe to execute. The role of the partial specification is to capture the desired static safety conditions (e.g., never collide) and provide a structure to combine scenario/mode analysis with action space

<sup>1</sup>Restriction to finite time is in theory negligible; but in practice, over-approximation limits guarantees to durations of at most seconds, which is insufficient to make meaningful conclusions about the safety of, e.g., a multi-hour flight.

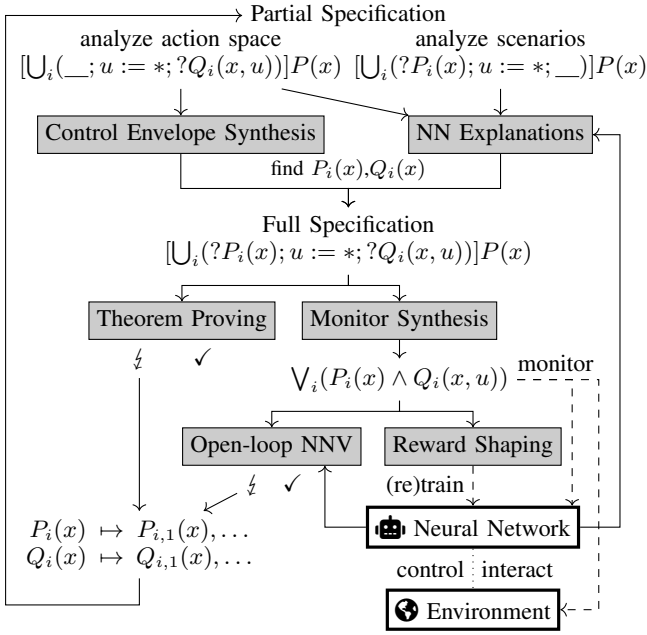


Figure 1. Overview of theorem proving, runtime monitoring, and neural network analysis in a uniform logic framework: a partial specification is filled in with control envelope synthesis or NN explanations; the resulting full specification is checked by a theorem prover, and used to derive monitors and shape rewards; open-loop NNV checks compliance with the specification.

analysis. Unlike other specification languages (e.g., signal temporal logic [24]), which require careful manual phrasing of predictive safety conditions, in dGL we separate the static correctness conditions from the predictive model; predictive conditions are then formed through proofs. The holes in the partial specification are anchor points for formal methods and neural network analysis to fill in their results and turn the partial specification into a full specification.

*a) From Partial to Full Specification:* Control envelope synthesis techniques and neural network analysis techniques interoperate to fill in a partial specification in the following analysis steps: (i) analyze scenarios by providing concrete descriptions of scenarios  $P_i(x)$  in the partial specifications (e.g., large distance to lead car) to fill in the neural network response  $Q_i(x, u)$  from neural network explanations; and (ii) analyze the action space by providing concrete descriptions of actions  $Q_i(x, u)$  in the partial specification (e.g., full deceleration) to fill in the scenarios from neural network input regions that elicit these responses, or from control envelope synthesis techniques that generate the largest safe envelopes.

*b) Full Specification Correctness:* When attempting to verify the correctness of the full specification, a theorem prover will detect whether the formal description of the neural network covers the full range of possible inputs/outputs, whether all responses of the neural network are correct, and whether it applies choices only in the appropriate scenarios. A successful proof certifies the neural network as correct; if incorrect, the theorem prover reports counterexamples. These counterexamples may then guide neural network retraining,

be split into smaller regions to try certify subregions, or be treated in the formal model itself to obtain a safety envelope to safeguard the neural network in its now known regions of incorrectness. This process benefits from the ability of neural network explanation tools (e.g., Fanoos [22]) to adjust the  $P_i(x)$  and  $Q_i(x)$  predicates to the wanted level of granularity in the verification and refinement loop of the process.

*c) Runtime Monitoring:* Offline safety proofs produce strong guarantees about system correctness under the assumption that the formal model and the training environment represent the operating environment well. To make this assumption explicit in the system, ModelPlex [21] turns formal models into runtime monitors that check the learning-enabled system at runtime for compliance with the safety proof; monitors are also suitable to check the environment for meeting the expectations set during training. A correctness proof links the offline proofs with the runtime observations of the monitor. A monitor acts as a switch to a correct fallback strategy in a Simplex-like [25] or shielding-like setup (e.g., [26]).

*d) Reward Shaping and Neural Network Verification:* Since extensive interference from monitors and fallback control is undesirable, the framework includes a reward shaping step [23] to turn formal models and monitors into reward function components to steer the training process itself and produce learned controllers that operate safely within the constraints of the formal model. Open-loop neural network verification on the basis of synthesized monitors [15] then proves compliance of the learned controller with the formal specification, or provides counterexample regions that can be combined with the synthesized runtime monitors to shield the system from executing unsafe actions.

#### A. Neural Network Control Envelopes

At an abstract level, a learned controller (neural network) can be summarized as a black-box function  $u = nn(x)$  from state  $x$  to control output  $u$ . The neural network interacts periodically with a continuous time-space environment model, possibly facing adversarial inputs. Such a setup can be expressed with a hybrid game model  $sys$  as below:

$$sys \equiv (u := nn(x); (v := *)^d; \\ t := 0; \{x' = f(x, u, v), t' = 1 \ \& \ t \leq T\})^* .$$

The model  $sys$  includes a differential equation model  $x' = f(x, u, v)$  of the environment response ( $x'$  denotes  $\frac{dx}{dt}$ ) and the adversarial inputs  $v := *$  where the operator  $\cdot^d$  represents adversarial choice. Interaction can be repeated ( $\cdot^*$ ) in a time-triggered way, i.e., a timer  $t$  executes the neural network with a latency of at most time  $T$ .

Including a neural network verbatim into a formal model typically makes infinite-time safety analysis infeasible. Instead, the formal model describes a control envelope  $nn(x) \equiv \bigcup_i (?P_i(x); u := *; ?Q_i(x, u))$  that defines expected outputs  $Q_i(x, u)$  in certain input regions  $P_i(x)$  (“what do you do when”) or, conversely, the required input to elicit certain output behavior (“when do you do what”), obtained e.g. by abstract interpretation [22]. Safety statements about model  $sys$  are

expressed in formulas of the shape  $Q(x) \rightarrow [sys]P(x)$ , which expresses that all runs of the game  $sys$  satisfy postcondition  $P(x)$  when started in states that satisfy initial conditions  $Q(x)$ . Stability [27] can be expressed and verified as well. The KeYmaera X theorem prover [17] provides a tactics language and library [28] to analyze such models. This allows separating closed-loop analysis in the theorem prover from open-loop neural network verification as an isolated outside step.

*a) Challenge—Extract a Control Envelope from a Neural Network:* Fanoos [22] analyzes neural networks for explanations “what do you do when” to compute  $Q_i(x, u)$  for given  $P_i(x)$  and “when do you do what” to obtain  $P_i(x)$  for given  $Q_i(x, u)$ . Proof-guided synthesis techniques [18] can provide reference conditions from an optimal model-predictive template in differential game logic: For example, a model-predictive condition  $Q_i(x, u) \equiv [(v := *)^d; x' = f(x, u, v)]P(x)$  expresses that choosing  $u$  guarantees  $P(x)$  when facing the adversarial and environment model. Finding, refining, and certifying such conditions requires tight integration between neural network analysis, control envelope synthesis, and theorem proving.

### B. Runtime Monitoring and Neural Network Verification

The differential equations and nondeterministic alternatives in hybrid programs make them an expressive specification language, but for execution require computationally expensive methods (e.g., online reachability analysis). Much of this computational complexity can be shifted offline through the use of ModelPlex [21]: ModelPlex formulas are quantifier- and modality-free, and are thus computationally inexpensive to evaluate from concrete measurements at runtime, which makes them attractive for reinforcement learning and neural network verification. Based on ModelPlex, [15] describes an approach for open-loop verification of neural networks; it includes a step to lift off-the-shelf neural network verification tools, which are typically limited to linear constraints, to the nonlinear constraints produced by ModelPlex. The predictive nature of ModelPlex monitors is vital for shielding learned controllers during training and during operation [3].

*a) Challenge—Sim-to-Real Transfer Monitoring:* Sim-to-real transfer in reinforcement learning (see [29] for a survey) typically addresses feature or efficiency loss of a trained controller. A complementary view to sim-to-real transfer focuses on detecting when the safety-relevant margins of the learned controller become unfavorably close to unsafety. Quantitative representations of ModelPlex safety monitors [23] can serve as a source for obtaining a safety signature in simulation for the purpose of using system identification to build a model of the safety signature across different scenarios. A difference between the safety margins in training and at runtime identifies potentially unknown operating conditions, deviation between the training environment and the true operating environment, and risk of safety degradation.

*b) Challenge—Critic Assessment:* Actor-critic learning separates the responsibility of choosing an action (the actor) from the responsibility of predicting the quality of the action

(the critic). This setup is especially well-suited for integration with the safety envelopes specified in formal hybrid systems models: [30] informally compares the conditions  $P_i(x)$  of the formal model and the predictions made by a critic. A formal foundation to characterize the difference between the conditions  $P_i(x)$  of the formal model and the predictions of the critic paves the way towards automated assessment. A challenge is determining whether the cause of a detected difference is an overly optimistic (unsafe) critic or an overly conservative (safe) control envelope in the formal model.

### C. Verification-Driven Reward Shaping

A hybrid systems model specifies both the control actions and the response of the environment; a monitor that is derived from the discrete  $u := mn(x)$  using the controller monitor configuration of ModelPlex [21] can be used to shield an unverified controller. A quantitative interpretation of such a monitor [23, Def. 2] measures the *predictive* safety margin of a control decision: negative safety margin means the action will inevitably result in future unsafety if it is not replaced with safe fallback control. Including the safety margin into a reward function gives predictive feedback without the necessity of executing until unsafety, which is important for safe online reinforcement learning. Safety-preserving signal rescaling [23] can have considerable influence on the learning process.

*a) Challenge—Liveness Rewards:* Liveness proofs certify that a system design enables reaching some goal (e.g., pass through an intersection with cross-traffic). A liveness proof must find either a finite strategy or a function that measures progress towards reaching the liveness goal. These proof insights may serve as artifacts in reinforcement learning: a finite strategy of control choices can be used as a starting point for imitation learning, whereas a progress function can be used as a component of the reward function in reinforcement learning (the learning agent gets reward for beating the liveness proof).

### ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. CCF2427581.

### REFERENCES

- [1] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang, and A. C. Knoll, “A review of safe reinforcement learning: Methods, theory and applications,” *CoRR*, vol. abs/2205.10330, 2022.
- [2] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, “Safe learning in robotics: From learning-based control to safe reinforcement learning,” *Annu. Rev. Control. Robotics Auton. Syst.*, vol. 5, pp. 411–444, 2022.
- [3] N. Fulton and A. Platzer, “Safe reinforcement learning via formal methods: Toward safe control through proof and learning,” in *AAAI*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 6485–6492.
- [4] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee, “Verifying the safety of autonomous

- systems with neural network controllers,” *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 1, pp. 7:1–7:26, 2021.
- [5] N. Hunt, N. Fulton, S. Magliacane, T. N. Hoang, S. Das, and A. Solar-Lezama, “Verifiably safe exploration for end-to-end reinforcement learning,” in *HSCC*, S. Bogomolov and R. M. Jungers, Eds. ACM, 2021, pp. 14:1–14:11.
- [6] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, “Safe reinforcement learning via shielding,” in *AAAI*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 2669–2678.
- [7] G. D. Giacomo, L. Iocchi, M. Favorito, and F. Patrizi, “Restraining bolts for reinforcement learning agents,” in *AAAI*. AAAI Press, 2020, pp. 13 659–13 662.
- [8] M. Hasanbeig, D. Kroening, and A. Abate, “LCRL: certified policy synthesis via logically-constrained reinforcement learning,” in *QEST*, ser. LNCS, E. Ábrahám and M. Paolieri, Eds., vol. 13479. Springer, 2022, pp. 217–231.
- [9] E. Yel, T. J. Carpenter, C. D. Franco, R. Ivanov, Y. Kantaros, I. Lee, J. Weimer, and N. Bezzo, “Assured runtime monitoring and planning: Toward verification of neural networks for safe autonomous operations,” *IEEE Robotics Autom. Mag.*, vol. 27, no. 2, pp. 102–116, 2020.
- [10] H. Tran, N. Pal, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, “Verification of piecewise deep neural networks: a star set approach with zonotope pre-filter,” *Formal Aspects Comput.*, vol. 33, no. 4-5, pp. 519–545, 2021.
- [11] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, “Reluplex: An efficient SMT solver for verifying deep neural networks,” in *CAV*, ser. LNCS, R. Majumdar and V. Kuncak, Eds., vol. 10426. Springer, 2017, pp. 97–117.
- [12] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. J. Kochenderfer, and C. W. Barrett, “The Marabou framework for verification and analysis of deep neural networks,” in *CAV*, ser. LNCS, I. Dillig and S. Tasiran, Eds., vol. 11561. Springer, 2019, pp. 443–452.
- [13] X. Yang, T. T. Johnson, H. Tran, T. Yamaguchi, B. Hoxha, and D. V. Prokhorov, “Reachability analysis of deep ReLU neural networks using facet-vertex incidence,” in *HSCC’21*, S. Bogomolov and R. M. Jungers, Eds. ACM, 2021, pp. 18:1–18:7.
- [14] Y. Y. Elboher, J. Gottschlich, and G. Katz, “An abstraction-based framework for neural network verification,” in *CAV*, ser. LNCS, S. K. Lahiri and C. Wang, Eds., vol. 12224. Springer, 2020, pp. 43–65.
- [15] S. Teuber, S. Mitsch, and A. Platzer, “Provably safe neural network controllers via differential dynamic logic,” in *NIPS*, A. Globerson, L. Mackey, A. Fan, C. Zhang, D. Belgrave, J. Tomczak, and U. Paquet, Eds. Curran Associates, Inc., 2024.
- [16] A. Platzer, “Differential game logic,” *ACM Trans. Comput. Log.*, vol. 17, no. 1, p. 1, 2015.
- [17] N. Fulton, S. Mitsch, J. Quesel, M. Völöp, and A. Platzer, “KeYmaera X: an axiomatic tactical theorem prover for hybrid systems,” in *CADE-25*, ser. LNCS, A. P. Felty and A. Middeldorp, Eds., vol. 9195. Springer, 2015, pp. 527–538.
- [18] A. Kabra, J. Laurent, S. Mitsch, and A. Platzer, “CESAR: control envelope synthesis via angelic refinements,” in *TACAS*, ser. LNCS, B. Finkbeiner and L. Kovács, Eds., vol. 14570. Springer, 2024, pp. 144–164.
- [19] A. Müller, S. Mitsch, W. Retschitzegger, W. Schwinger, and A. Platzer, “Tactical contract composition for hybrid system component verification,” *Int. J. Softw. Tools Technol. Transf.*, vol. 20, no. 6, pp. 615–643, 2018.
- [20] M. Brieger, S. Mitsch, and A. Platzer, “Uniform substitution for dynamic logic with communicating hybrid programs,” in *CADE*, ser. LNCS, B. Pientka and C. Tinelli, Eds., vol. 14132. Springer, 2023, pp. 96–115.
- [21] S. Mitsch and A. Platzer, “Modelplex: verified runtime validation of verified cyber-physical system models,” *Formal Methods Syst. Des.*, vol. 49, no. 1-2, pp. 33–74, 2016.
- [22] D. Bayani and S. Mitsch, “Fanoos: Multi-resolution, multi-strength, interactive explanations for learned systems,” in *VMCAI*, ser. LNCS, B. Finkbeiner and T. Wies, Eds., vol. 13182. Springer, 2022, pp. 43–68.
- [23] M. Qian and S. Mitsch, “Reward shaping from hybrid systems models in reinforcement learning,” in *NFM*, ser. LNCS, K. Y. Rozier and S. Chaudhuri, Eds., vol. 13903. Springer, 2023, pp. 122–139.
- [24] O. Maler and D. Nickovic, “Monitoring temporal properties of continuous signals,” in *FORMATS*, ser. LNCS, Y. Lakhnech and S. Yovine, Eds., vol. 3253. Springer, 2004, pp. 152–166.
- [25] L. Sha, “Using simplicity to control complexity,” *IEEE Softw.*, vol. 18, no. 4, pp. 20–28, 2001.
- [26] B. Könighofer, J. Rudolf, A. Palmisano, M. Tappler, and R. Bloem, “Online shielding for reinforcement learning,” *Innov. Syst. Softw. Eng.*, vol. 19, no. 4, pp. 379–394, 2023.
- [27] Y. K. Tan, S. Mitsch, and A. Platzer, “Verifying switched system stability with logic,” in *HSCC*, E. Bartocci and S. Putot, Eds. ACM, 2022, pp. 2:1–2:11.
- [28] N. Fulton, S. Mitsch, R. Bohrer, and A. Platzer, “Bellerophon: Tactical theorem proving for hybrid systems,” in *ITP*, ser. LNCS, M. Ayala-Rincón and C. A. Muñoz, Eds., vol. 10499. Springer, 2017, pp. 207–224.
- [29] W. Zhao, J. P. Queralta, and T. Westerlund, “Sim-to-real transfer in deep reinforcement learning for robotics: a survey,” in *2020 IEEE Symposium Series on Computational Intelligence, SSCI 2020, Canberra, Australia, December 1-4, 2020*. IEEE, 2020, pp. 737–744.
- [30] A. Kopylov, S. Mitsch, A. Nogin, and M. A. Warren, “Formally verified safety net for waypoint navigation neural network controllers,” in *FM*, ser. LNCS, M. Huisman, C. S. Pasareanu, and N. Zhan, Eds., vol. 13047. Springer, 2021, pp. 122–141.