Research Statement

Osbert Bastani

1 Introduction

Machine learning is revolutionizing the way we develop software. On the one hand, machine learning models such as deep neural networks (DNNs) are increasingly being incorporated into software, forming *neurosymbolic systems*. These programs are arising in safety-critical applications such as robotics, as well as human-AI systems such as healthcare and judicial decision-making. Simultaneously, programming assistants based primarily on large language models (LLMs) are becoming ubiquitous, helping programmers write code, find bugs, and generate test suites. **The goal of my research is to design novel tools to make it easier to develop software in the age of machine learning.** It can be organized into three directions:

- **Trustworthy neurosymbolic systems:** How can we ensure that neurosymbolic systems are trustworthy when they include intrinsically fallible components (i.e., machine learning models)? We have designed novel techniques that can provide *probabilistic* guarantees for neurosymbolic systems. Our techniques leverage uncertainty quantification to provide probabilistic guarantees for individual machine learning components; then, they compose them to obtain probabilistic guarantees for the overall system.
- **Synthesizing neurosymbolic programs:** Neurosymbolic programs can be significantly harder to develop compared to traditional programs due to the fuzzy nature of machine learning models. How can we make it easier to write neurosymbolic programs? Unlike traditional programming, where the specification is binary (either satisfied or not), in neurosymbolic programming, the specification is typically an objective function (e.g., accuracy). We have developed novel optimal synthesis algorithms that efficiently traverse the search space to identify the neurosymbolic program that optimizes a user-provided objective.
- **Machine learning for programmer productivity:** How can machine learning improve programmer productivity? How can we do so while ensuring the resulting code is trustworthy? We have leveraged machine learning in a number of programmer productivity tools including program synthesis, optimization, verification, and testing. In addition, we have worked on tools for rigorously quantifying the uncertainty of code completion models.

2 Trustworthy Neurosymbolic Systems

A key challenge to building trustworthy systems out of machine learning components is that these components are naturally error prone—e.g., we can never guarantee that a DNN object detector correctly detects every obstacle in front of a robot. Uncertainty quantification is a promising solution: if a machine learning component is uncertain, then the system can act more conservatively to maintain safety. For instance, a robot can act conservatively when the DNN is unsure if there

is an obstacle. We have developed a number of techniques for uncertainty quantification that come with probabilistic correctness guarantees. Typical uncertainty quantification techniques make the i.i.d. assumption that the training and test distributions are identical (or the closely related exchangeability assumption); thus, we have also designed techniques for detecting and mitigating the impact of distribution shift. Finally, we have designed algorithms inspired by these techniques for offline reinforcement learning, which can enable learning in a safe way from large-scale batch data without requiring potentially dangerous interactions with the environment.

Conformal prediction for neurosymbolic systems. Conformal prediction is a collection of techniques from statistics for quantifying predictive uncertainty by modifying an underlying model to output sets of labels instead of individual labels [1]. These algorithms come with *coverage guarantees*—in particular, under the assumption that the training and test distribution are identical, the prediction sets are guaranteed to contain the ground truth label with high probability. For building trustworthy neurosymbolic programs, conformal prediction has several advantages compared to more traditional techniques for uncertainty quantification that predict a probability for each label (e.g., calibrated prediction). First, prediction sets tend to be easier to incorporate into existing software compared to probabilities (e.g., a traditional robot planning algorithm can avoiding prediction sets of obstacles, whereas planning under probabilistic predictions requires modifying the planning algorithm). In addition, the coverage guarantee often translates directly to safety guarantees for the overall system (e.g., a robot may be guaranteed to avoid obstacles with high probability), whereas guarantees provided for predicted probabilities are less interpretable.

Our recent work has demonstrated how techniques from learning theory can be used to design conformal prediction algorithms that come with probably approximately correct (PAC) guarantees [2]. Our work was also the first to demonstrate how conformal prediction can be applied to deep neural networks including ResNet [3], and there has been a great deal of subsequent interest in combining conformal prediction and deep learning [4, 5]. In follow up work, we have demonstrated how coverage guarantees provided by conformal prediction can be used to provide probabilistic guarantees for broader neurosymbolic programs. For instance, we have combined it with model predictive shielding—a safe reinforcement learning algorithm we developed in our prior work [6, 7, 8]—to enable safe reinforcement learning from visual observations, where a reinforcement learning agent uses a DNN policy to directly map images to control inputs [9]. In another line of work, we have shown how to build conformal prediction sets for large language models and compose them to provide probabilistic guarantees for retrieval augmented question answering [10]. Finally, in ongoing work, we have worked on extending these techniques to general program compositions by using abstract interpretation to propagate prediction sets through the program [11].

Uncertainty quantification under distribution shift. Traditional algorithms for uncertainty quantification, including both conformal and calibrated prediction, rely heavily on the i.i.d. assumption that the training and test distributions are identical (or the slightly weaker exchangeability assumption). In many real-world settings, these assumptions break down—e.g., a robot operating in a changing environment or a machine learning model deployed in a new hospital with a different patient population. Thus, in addition to quantifying uncertainty under the i.i.d. assumption, we need to detect when this assumption fails. We consider the unsupervised domain adaptation setting (i.e., we have unlabeled examples from a shifted test distribution), which holds in many settings since the system can observe the inputs for which it needs predicted outputs. Then, we propose to use *classifier based tests* to detect shifts in the covariate distribution [12]. Intuitively, the idea is to train a DNN to distinguish the training and test inputs. If they can be distinguished, then a

shift has definitely occurred; for instance, if we can reliably predict whether an image came from a robot's training or test environment, then the two environments cannot be identically distributed. While the converse is not true due to limitations of the DNN (i.e., there may be a shift even if the DNN does not detect one), the effectiveness of DNNs at prediction minimizes this risk.

We can also use the DNN classifier for uncertainty quantification under distribution shift. Intuitively, the DNN prediction probability can be used to derive *importance weights*, which quantify the degree of shift for individual inputs. These weights can be used to adjust uncertainty quantification to match the shifted test distribution. We have proposed both calibrated [13] and conformal [14] prediction algorithms for the covariate shift setting. In addition, we have proposed conformal prediction algorithms for the label shift setting [15], the meta-learning setting [16], and the online adversarial setting [17]. These algorithms can be combined with the approaches described above to provide guarantees for neurosymbolic programs under distribution shift.

Safe offline reinforcement learning. While distribution shift affects many real-world systems, it is a particularly important challenge in reinforcement learning, where changes to the control policy naturally cause a shift in the distribution of visited states. These challenges are particularly pertinent in *offline reinforcement learning*, where the goal is to learn from a large-scale offline dataset (e.g., collected by observing humans completing tasks in the world) without any online interaction. One way to mitigate distribution shift in offline reinforcement learning is to modify the objective to ensure the agent visits a similar state distribution as the offline dataset—e.g., using a KL divergence constraint. We have demonstrated how to extend this strategy to learning from visual observations in the goal-conditioned setting [18, 19], a prerequisite for training foundation models for control. When applied to large-scale human video data, these techniques have produced foundation models capable of solving novel tasks from minimal specifications in the form of either a handful of demonstrations [20] or natural language [21]. For example, these techniques have been applied to solving tasks such as folding cloth or grasping complicated objects from just a few demonstrations. We have also shown how long-horizon tasks can be automatically decomposed into shorter tasks, which can then be solved using policies trained based on our foundation models [22]. Finally, we have demonstrated how offline reinforcement learning can be used as a fundamental building block to enable reinforcement learning with safe exploration constraints. Intuitively, we can iteratively use the current policy to safely collect data, and then train a new safe policy based on the new data. Our approach comes with safety guarantees in the finite state setting [23], and performs well empirically for continuous state and action spaces [24].

3 Synthesizing Neurosymbolic Programs

In traditional programming, there is typically a notion of correctness that the program should satisfy (which can be formally encoded as a logical specification). In contrast, in neurosymbolic programming, the system may make mistakes due to errors in the machine learning components. Thus, the specification must somehow account for the possibility for errors. A standard way to do so is to instead provided a quantitative objective such as the accuracy that the program should optimize. However, optimizing such objectives tends to be challenging for programmers, since they need to understand the performance properties of individual machine learning components as well as how different components may interact with one another. As a consequence, there is an important need for algorithms to automatically synthesize neurosymbolic programs. We propose neurosymbolic synthesis algorithms for various settings. First, focusing on neurosymbolic programs designed

to query unstructured data, we propose efficient synthesis algorithms targeting the programming-by-examples setting that are guaranteed to return the optimal program on the given examples. In addition, we have proposed techniques for adapting these techniques to synthesizing neurosymbolic programs that implement control policies in the reinforcement learning setting.

Querying unstructured data. One important category of neurosymbolic programs arises when users seek to write queries over unstructured data such as text or images. In these cases, a natural strategy is to use machine learning components to predict structure in the data (e.g., entity recognition, object detection, etc.), and then query the resulting structure using traditional components. We have designed algorithms for automatically synthesizing queries over unstructured data in several domains, including extracting information from websites, where LLMs are used to match keywords of interest [25], and identifying events in video camera data, where object trackers are used to identify trajectories in the data [26, 27]. For instance, our algorithms might be used to help a data scientist automatically identify intersections where pedestrians frequently cross the street. These algorithms provably find the program that optimize a user-provided objective.

Subsequently, we have designed a general-purpose algorithm for efficiently synthesizing optimal programs that can be applied to queries in a given domain-specific language (DSL) [28]. This work builds on our prior work on program synthesis [29], where we use deductive reasoning to prove that certain branches of the search tree of programs are infeasible. In optimal synthesis, we instead use abstract interpretation to compute upper bounds on the objective value of a given branch of the search space; these upper bounds can similarly be used to prune the search space if a program exceeding that upper bound has already been identified. In our neurosymbolic setting, this approach is two orders of magnitude faster than SMT-based approaches to optimal synthesis.

Program synthesis for reinforcement learning. Another setting where neurosymbolic programming can be an effective strategy is reinforcement learning. In particular, our work has demonstrated that by training control policies in the form of programs instead of end-to-end neural networks, we can obtain policies that are more interpretable [30], generalize more robustly to out-of-distribution scenarios [31], and are easier to formally verify [32, 33]. For instance, we have trained a provably correct decision tree policy for a toy game of Pong [32], which to the best of our knowledge is the first formally verified closed-loop guarantee for a learned policy. A particular challenge for formal verification in control is that the environment itself often differs from one problem instance to another. For example, we may want a control policy that can drive safely on a variety of different road configurations. Our work has demonstrated how neurosymbolic policies can solve this problem [33]. We have designed policies that are composed from primitive neural policies in a way that mirrors the geometry of the road; in particular, each neural policy is associated with a road segment such as a straight or a turn. Furthermore, we have proven that each neural policy solves its portion of the road in a way that is compatible across different policies. Thus, our compositional policy can provably drive safely on any road geometry obtained by composing these primitive pieces. This strategy paves the way for formal methods to apply to guaranteeing safety in variable environments, a key challenge for formally verified reinforcement learning.

Programs have also proven effective for reinforcement learning in other ways. For instance, our recent work has demonstrated how large language models (LLMs) can be leveraged to synthesize reward functions for reinforcement learning. In particular, reward functions are typically implemented as Python code, so we may naturally expect LLMs for code to be able to generate reward functions. Our approach additionally leverages feedback from the simulator to iteratively improve the given reward function. We demonstrate that our strategy outperforms human-written reward

functions on a benchmark of dexterous manipulation tasks [34], and have additionally demonstrated how it can be used to train a real-world robot dog (in particular, the Unitree Go1) to walk [35].

Specification-guided reinforcement learning. In many scenarios, the user has prior knowledge of the high-level structure of the task they would like to solve. We have developed a specification language enabling users to provide a formal specification encoding this structure, which helps guide the reinforcement learning algorithm to solve the task more effectively [36]. Then, we can use reinforcement learning to automatically train low-level control policies that implement primitives in the specification. For instance, the specification might indicate a sequence of subtasks the robot needs to complete, in which case our framework would train a neural policy to solve each subtask. In addition, we have designed an algorithm for training these policies in an effective way by interweaving high-level planning and low-level reinforcement learning [37, 38]. By leveraging the high-level structure as guidance, we can efficiently learn to solve long-horizon tasks that are challenging for traditional, end-to-end reinforcement learning. Our experiments demonstrate that our approach can achieve significantly better performance at complex, long-horizon grasping and navigation tasks compared to state-of-the-art end-to-end reinforcement learning algorithms.

4 Machine Learning for Programmer Productivity

We have worked on using machine learning to improve programmer productivity tools, including program synthesizers [29, 39], program optimizers [40], program verifiers [41], and fuzz testers [42]. In particular, machine learning is incorporated to guide search over a large space—e.g., programs in program synthesis, proof strategies in program verification, and inputs in fuzz testing.

Reinforcement learning for program synthesis. There has been a great deal of success in using machine learning to improve program synthesis. The basic strategy is to use a pretrained machine learning model to determine the search order—e.g., when searching over a space of possible programs given by a context-free grammar, we can use the model to choose the most likely production to use to expand a nonterminal in the search tree. However, this strategy has the shortcoming that the model cannot be adapted to features of the current problem. Deep reinforcement learning is a natural solution to this problem: thinking of the machine learning model as a control policy (i.e., deciding what branch of the search space to try next), we can use feedback from the search procedure to generate policy gradient updates to the machine learning model, enabling it to learn strategies specialized to the current program. While this strategy works well, we can further improve performance by more tightly coupling the symbolic search with the reinforcement learning algorithm. In particular, symbolic search typically uses deductive reasoning (e.g., an SMT solver) to help prune infeasible branches of the search space. We have demonstrated how feedback from deductive reasoning (e.g., the UNSAT core produced by an SMT solver) can be used to improve the policy gradient update [39]. In particular, we can incorporate any additional programs ruled out by deductive reasoning as off-policy information in the policy gradient update, enabling the machine learning model to learn to reduce the likelihood of these kinds of programs as well. Our experiments show that our strategy improves synthesis time by 8.71× compared to prior work.

Large language models for program optimization. More recently, large language models (LLMs) have significantly advanced the capabilities of machine learning in the code domain by enabling them to understand complex program logic. In our recent work on program optimization, we have benchmarked various techniques for leveraging LLMs to optimize programs [40]. The basic idea is to give the LLM a program as input, and ask it to produce an optimized version

of that program as output. This can be achieved using either blackbox prompting techniques (e.g., in-context learning, chain-of-thought prompting, etc.), or by finetuning the model. We found *dynamic in-context learning* (i.e., retrieve similar examples) to be especially effective in the blackbox setting. In the finetuning setting, one surprising challenge is that naïvely training on slow-fast program pairs does not produce very good results. Intuitively, the reason is that the model cannot distinguish between "bad" vs. "good" slow-fast pairs (i.e., ones with a small vs. large speedup). One simple solution is to train only on slow-fast pairs with a large speedup. However, we found that a more effective solution is to actually provide the speedup as part of the input to the model—i.e., the input is the pair of the original program and the speedup, and the output is the optimized program. Intuitively, by conditioning on the speedup achieved, the model can determine which kinds of changes lead to what level of speedup. Then, during inference, we ask the model for the largest speedup possible. For CodeLlama-13B, our approach produced an average speedup of 5.65× compared to the baseline speedup of 2.71×, demonstrating the effectiveness of this approach.

Uncertainty quantification for code generation. One of the main challenges using LLMs for code generation is the lack of trust in the generated code. We have proposed to use uncertainty quantification to mitigate this problem. A key issue because code is a structured output, naïve prediction sets have the potential to be very large. To address this issue, we have designed algorithms for inferring prediction sets that can be represented in a compact, structured form [43]. For example, for code generation, a prediction set can be represented by a partial program, which implicitly represents the set of all completions of that partial program. Even if this set is large, the partial program provides an interpretable representation of uncertainty in the predicted program.

5 Conclusions and Future Work

My research has laid the foundation for trustworthy neurosymbolic programming by developing uncertainty quantification as a tool to specify probabilistic correctness for machine learning components. These individual guarantees can then be composed to form guarantees for the overall program. Simultaneously, my work on optimal program synthesis has automated the development of neurosymbolic programs, while providing rigorous guarantees on the optimality of the resulting program. These techniques can significantly improve our ability to build trustworthy neurosymbolic programs as machine learning components become increasingly pervasive in real-world systems.

My ongoing work on neurosymbolic programming aims to make it easier to develop neurosymbolic programs in other ways. One key challenge is that while neurosymbolic programs tend to be highly parallelizable, actually exposing this parallelism can be quite challenging when they are written in general purpose programming languages such as Python. We are working on a novel language based on lambda calculus for writing neurosymbolic programs [44]. Because this language is pure by design, it can automatically be evaluated in a fully parallel way. We show that rewriting programs in our language can improve performance by an order of magnitude—e.g., our implementation of tree-of-thought prompting runs 4.8× speedup compared to the original implementation.

We are also working on better foundation models for code, which can improve the usability of machine learning across the software development pipeline. Building on our work on performance conditioned finetuning for program optimization, we are training models that are conditioned on a large variety of information about the program—e.g., the libraries it uses, the syntactic expressions used, measures of code quality, etc. Our early results suggest that these kinds of models can provide programmers with significantly greater control over the constructs used in the generated code.

References

[1] Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer, 2005.

[2] Sangdon Park, Osbert Bastani, Nikolai Matni, and Insup Lee. PAC confidence sets for deep neural networks via calibrated prediction. *The International Conference on Learning Representations (ICLR)*, 2020.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] Anastasios Nikolas Angelopoulos, Stephen Bates, Michael Jordan, and Jitendra Malik. Uncertainty sets for image classifiers using conformal prediction. *The International Conference on Learning Representations (ICLR)*, 2020.

[5] Anastasios N Angelopoulos and Stephen Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *arXiv preprint arXiv:2107.07511*, 2021.

[6] Shuo Li and Osbert Bastani. Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[7] Osbert Bastani. Safe reinforcement learning with nonlinear dynamics via model predictive shielding. *American Control Conference (ACC)*, 2021.

[8] Osbert Bastani, Shuo Li, and Anton Xue. Safe reinforcement learning via statistical model predictive shielding. *Robotics: Science and Systems (RSS)*, 2021.

[9] Sangdon Park, Shuo Li, Insup Lee, and Osbert Bastani. PAC confidence predictions for deep neural network classifiers. *The International Conference on Learning Representations (ICLR)*, 2021.

[10] Shuo Li, Sangdon Park, Insup Lee, and Osbert Bastani. Traq: Trustworthy retrieval augmented question answering via conformal prediction. *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024.

[11] Ramya Ramalingam, Sangdon Park, and Osbert Bastani. Uncertainty quantification for neurosymbolic programs via compositional conformal prediction. *arXiv preprint arXiv:2405.15912*, 2024.

[12] Sooyong Jang, Sangdon Park, Insup Lee, and Osbert Bastani. Sequential covariate shift detection using classifier two-sample tests. *The International Conference on Machine Learning (ICML)*, 2022.

[13] Sangdon Park, Osbert Bastani, Jim Weimer, and Insup Lee. Calibrated prediction with covariate shift via unsupervised domain adaptation. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[14] Sangdon Park, Edgar Dobriban, Insup Lee, and Osbert Bastani. PAC prediction sets under covariate shift. *The International Conference on Learning Representations (ICLR)*, 2022.

[15] Wenwen Si, Sangdon Park, Insup Lee, Edgar Dobriban, and Osbert Bastani. Pac prediction sets under label shift. *The International Conference on Learning Representations (ICLR)*, 2024.

[16] Sangdon Park, Edgar Dobriban, Insup Lee, and Osbert Bastani. Pac prediction sets for meta-learning. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[17] Osbert Bastani, Varun Gupta, Christopher Jung, Georgy Noarov, Ramya Ramalingam, and Aaron Roth. Practical adversarial multivalid conformal prediction. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[18] Jason Yecheng Ma, Jason Yan, Dinesh Jayaraman, and Osbert Bastani. Offline goal-conditioned reinforcement learning via f-advantage regression. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[19] Jason Yecheng Ma, Andrew Shen, Dinesh Jayaraman, and Osbert Bastani. Smodice: Versatile offline imitation learning via state occupancy matching. *The International Conference on Machine Learning (ICML)*, 2022.

[20] Jason Yecheng Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. VIP: Towards universal visual reward and representation via value-implicit pre-training. *The International Conference on Learning Representations (ICLR)*, 2023.

[21] Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. LIV: Language-image representations and rewards for robotic control. *The International Conference on Machine Learning (ICML)*, 2023.

[22] Charles Zhang, Yunshuang Li, Osbert Bastani, Abhishek Gupta, Dinesh Jayaraman, Jason Ma, and Lucas Weihs. Universal visual decomposer: Long-horizon manipulation made easy. *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[23] Wanqiao Xu, Jason Yecheng Ma, Kan Xu, Hamsa Bastani, and Osbert Bastani. Uniformly conservative exploration in reinforcement learning. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023.

[24] Jason Y. Ma, Dinesh Jayaraman, and Osbert Bastani. Conservative offline distributional reinforcement learning. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[25] Jocelyn Chen, Aaron Lamoreaux, Xinyu Wang, Greg Durrett, Osbert Bastani, and Isil Dillig. Web question answering with neurosymbolic program synthesis. *ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, 2021.

[26] Favyen Bastani, Songtao He, Ziwen Jiang, Osbert Bastani, and Sam Madden. Skyquery: An aerial drone video sensing platform. *Symposium on New Ideas in Programming and Reflections on Software (Onward!)*, 2021.

[27] Stephen Mell, Favyen Bastani, Steve Zdancewic, and Osbert Bastani. Synthesizing trajectory queries from examples. *International Conference on Computer Aided Verification (CAV)*, 2023.

[28] Stephen Mell, Steve Zdancewic, and Osbert Bastani. Optimal program synthesis via abstract interpretation. *ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2024.

[29] Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. Program synthesis using conflict-driven learning. *ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, 2018.

[30] Jeevana P. Inala, Yichen Yang, James Paulos, Yewen Pu, Osbert Bastani, Vijay Kumar, Martin Rinard, and Armando Solar-Lezama. Neurosymbolic transformers for multi-agent communication. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[31] Jeevana P. Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. *The International Conference on Learning Representations (ICLR)*, 2020.

[32] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[33] Radoslav Ivanov, Kishor Jothimurugan, Steve Hsu, Vaidya Shaan, Rajeev Alur, and Osbert Bastani. Compositional learning and verification of neural network controllers. *ACM SIGBED International Conference on Embedded Software (EMSOFT)*, 2021.

[34] Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *The International Conference on Learning Representations (ICLR)*, 2024.

[35] Jason Ma, William Liang, Hung-Ju Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. Language model guided sim-to-real transfer. *Robotics: Science and Systems (RSS)*, 2024.

[36] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. Composable specifications for reinforcement learning. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[37] Kishor Jothimurugan, Osbert Bastani, and Rajeev Alur. Abstract value iteration for hierarchical deep reinforcement learning. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

[38] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[39] Yanju Chen, Chenglong Wang, Osbert Bastani, Isil Dillig, and Yu Feng. Program synthesis using deduction-guided reinforcement learning. *International Conference on Computer Aided Verification (CAV)*, 2020.

[40] Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob Gardner, Yiming Yang, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. Learning performance-improving code edits. *The International Conference on Learning Representations (ICLR)*, 2024.

[41] Jai Chen, Jiayi Wei, Yu Feng, Osbert Bastani, and Isil Dillig. Relational verification using reinforcement learning. *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2019.

[42] Zhengkai We, Evan Johnson, Wei Yang, Osbert Bastani, Dawn Song, Jian Peng, and Tao Xie. Reinam: Reinforcement learning for input-grammar inference. *ACM International Conference on the Foundations of Software Engineering (FSE)*, 2019.

[43] Adam Khakhar, Stephen Mell, and Osbert Bastani. PAC prediction sets for large language models of code. *The International Conference on Machine Learning (ICML)*, 2023.

[44] Stephen Mell, Steve Zdancewic, and Osbert Bastani. An opportunistically parallel lambda calculus for performant composition of large language models. *arXiv preprint arXiv:2405.11361*, 2024.