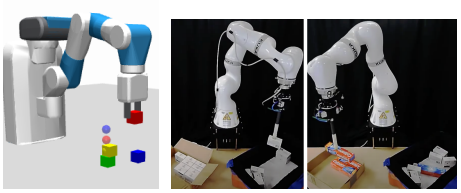
He Zhu's Research Statement

Overview. My research is driven by the belief that principles from programming languages (PL) and formal methods (FM) can help overcome key challenges in building reliable, learning-enabled systems. For decades, these principles have enabled developers to reason about, verify, and refine software through high-level abstractions and symbolic analysis. I aim to bring the same level of structure and trust to machine learning (ML). Achieving this requires a systematic integration of symbolic reasoning with statistical learning. My work focuses on forging this connection with the goal of building learning systems that are not only powerful but also interpretable and verifiable.

My current efforts are situated within the emerging field of neurosymbolic programming (NSP) (Chaudhuri et al., 2021), which bridges deep learning and program synthesis. At its core, NSP reframes learning as a synthesis task: the goal is to discover a program—expressed in a domain-specific language (DSL)—that optimizes a quantitative objective while satisfying additional semantic constraints (e.g. formal correctness guarantees). What draws me to this area is its unique ability to combine the rigor of FM with the flexibility of modern ML. I see NSP as a promising foundation for building trustworthy and efficient models. Its symbolic representations enable greater interpretability and verifiability than deep neural networks. Using programming abstractions, NSP also facilitates more data-efficient learning and robust generalization.

Motivation. Despite its promise, NSP presents significant challenges. First, NSP relies on DSLs to encode prior knowledge in symbolic form. Designing appropriate DSLs requires expert insight—deciding what symbolic elements (i.e. "symbols") to include and how to structure them is application-specific and often not easily scalable. In many settings, the semantics of neural components may themselves be learned as it is often unclear what exactly the neural modules should learn, or how their behavior should compose with the "symbols". Second, NSP combines training neural components and searching over discrete program structures, resulting in a vast combinatorial search space that is difficult to navigate. Lastly, while learning-enabled systems are increasingly used in safety-critical domains, formally guaranteeing the correctness of NSP systems is difficult, as it requires reasoning about discrete symbolic programs, the continuous behavior of neural modules, and the hierarchical composition of subprograms and learned modules. For example, consider the robotic tasks illustrated in Fig. 1. In the towering scenario, the robot must stack multiple cubes scattered on a table into a tower. Useful "symbols" might include state abstraction predicates such as whether a cube is currently grasped by the robot's gripper or whether it has been placed in its designated goal region (indicated by colored spheres), and abstract actions that involve high-level subgoals such as "grasp a cube" or "place a cube at the its goal region". These abstract actions must ultimately be grounded by neural controllers that operate in the low-level, continuous state space of the robot. Designing such symbolic abstractions a priori within a DSL demands significant human effort and domain expertise, limiting the scalability and generalizability of NSP. Even with well-crafted symbolic definitions, the program synthesis problem remains challenging due to the vast search space. For towering, based on the symbolic abstraction given above, the system must synthesize *loop* programs to generalize to an arbitrary number of cubes, with the loop conditional specifying the correct sequence in which the cubes should be stacked. These challenges are amplified in more complex tasks. For example, in the packing scenario in Fig. 1, the robot must identify and extract objects from an unstructured bin and then pack them tightly into a shipping box according to a predefined spatial configuration. This task requires careful structuring of the symbolic representations over both geometric constraints and object/action dependencies. These intertwined challenges—symbolic abstraction discovery, the division of labor between symbolic and neural components, and the difficulty of formally reasoning about NSP systems that combine discrete and continuous behaviors—highlight the need for principled approaches to guide the construction of NSP systems.



(a) `Towering` (b) `Packing`

Figure 1. Robotics Tasks.

My effort. In recent years, I have explored NSP techniques for building autonomous agents—particularly robotic systems that integrate learned control policies for decision making in high-dimensional, continuous, and uncertain environments,

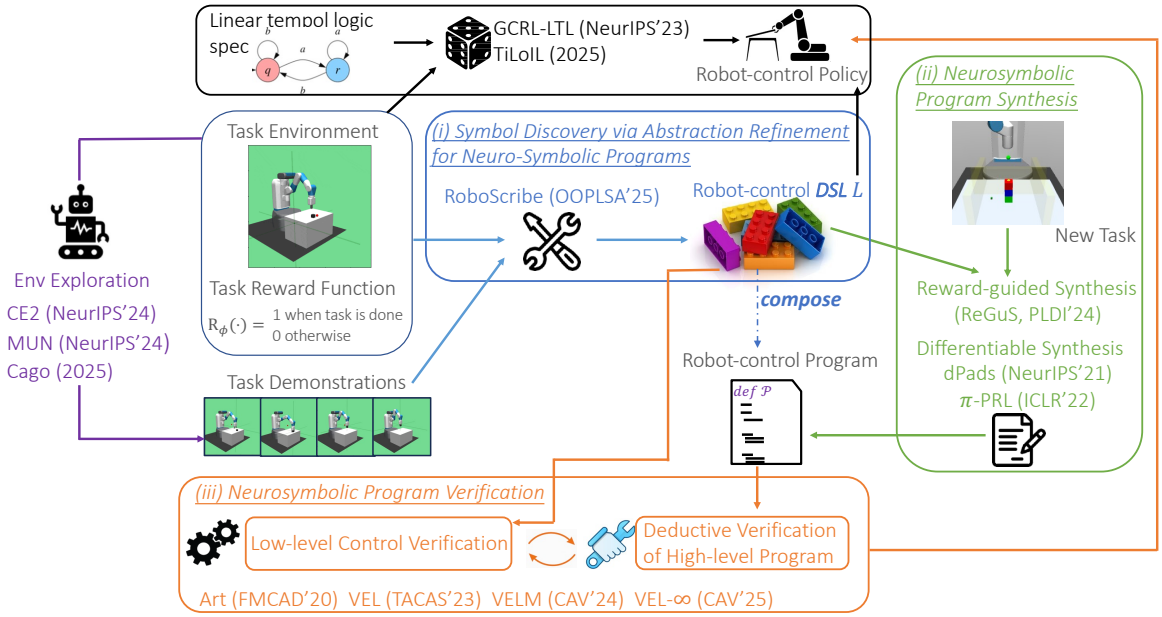


Figure 2. An overview of my work in the past 5 years.

such as those illustrated in Fig. 1. Safety, interpretability, and sample efficiency are essential for these systems. A promising direction in this space is programmatic reinforcement learning (Verma et al., 2018; Inala et al., 2020; Patton et al., 2024), where policies are expressed as domain-specific programs to improve transparency and reliability. However, existing approaches face the fundamental challenges of NSP mentioned above—namely, (1) designing suitable DSLs to encode domain knowledge in symbolic form, (2) navigating a combinatorially large program search space, and (3) ensuring the reliability of neurosymbolic programs. My research, summarized in Fig. 2, addresses these issues through new algorithmic techniques that make NSP more tractable and effective in the robotics domain. Specifically, I have developed (i) an abstraction refinement framework that automatically discovers symbolic definitions for neuro-symbolic robot-control programs (blue box in Fig. 2), addressing challenge (1); (ii) efficient neuro-symbolic synthesis algorithms that leverage reward signals to guide the search and mitigate the exponential complexity of the program space (green box in Fig. 2), addressing challenge (2); and (iii) verification techniques for neuro-symbolic programs, including deductive reasoning over high-level symbolic abstractions and modular verification of low-level control policies against their symbolic specifications in continuous state spaces (orange box in Fig. 2), addressing challenge (3).

Below, I outline a few core contributions that tackle these challenges head-on.

1. Neurosymbolic Robot-control Program Synthesis

My initial work on NSP for robot learning focuses on efficient synthesis algorithms given a well-designed neuro-symbolic DSL. This DSL combines standard language constructs with a library of neural components, each representing a task-agnostic skill that can be executed by a robot. Programs in this space are well-suited for complex, long-horizon robot-control tasks due to their capabilities for logical deduction and generalization—areas where deep reinforcement learning often falls short due to the lack of inductive biases in their learning representation.

1.1 Differentiable Synthesis. Unlike training deep neural nets, synthesizing programs requires optimizing over a combinatorial, non-differentiable, and rapidly exploded space of program structures. Even with strong heuristics, explicitly enumerating program structures is not scalable—especially when programs include parameters, such as weights in programmatic models, which must be learned separately for each candidate to fit the data. My research asks: *can we jointly search discrete program structures and program parameters using efficient optimization-based techniques*?

Our method, π-PRL (Qiu & Zhu, 2022), views program derivation as drawing programs from a probabilistic distribution (Cui & Zhu, 2021). Under this probabilistic view, we encode discrete program structure search as learning the probability distribution of the context-free grammar rules within a DSL from data to induce high-accuracy programmatic models. To this end, we express the program derivation process (i.e. iteratively applying grammar rules for program generation) as a

differentiable program. That is, on each partial program expressible in the search space, the categorical choice between applicable grammar rules to complete this program is relaxed into a softmax with trainable weights. The gradient updates for these softmax weights as well as any program parameters are guided by training data. For robotics, π-PRL (Qiu & Zhu, 2022) jointly learns control policy structures and parameters in a continuous relaxation of the program search space via an RL algorithm. It is capable of composing task-agnostic robot skills into an interpretable composite program to solve novel tasks. We have scaled this synthesis algorithm beyond robotics to learn functional programs with higher-order combinators for behavior classification (Cui & Zhu, 2021), and logical programs for recommendation tasks (Chen et al., 2022a;b).

1.2 Reward-guided Synthesis. Robotic tasks typically involve a long horizon, extending over thousands of control steps, and are characterized by sparse rewards, where reward signals are given only upon successful task completion or sporadically for critical steps. For example, in the long-horizon task `cleanhouse` depicted in Fig. 3, the agent is only rewarded for the percentage of wall-side trash bins picked up by the end of each episode. RL methods struggle with such long-horizon, sparse-reward tasks due to the exponential growth in required robot-environment interactions as the number of steps to discover rewards increases. Existing program synthesis approaches for RL, including π-PRL (Sec 1.1), inherit this limitation as they either adapt standard RL algorithms to guide program search or synthesize control programs by imitating RL models. My research investigates: *can we use the inductive biases in language constructs such as state-conditioned loops to effectively compress the exploration space for long-horizon, multi-stage, and procedural robot-control tasks?*

We developed ReGuS (Cui et al., 2024), a robot-control program synthesis algorithm based on DSLs with *user-defined* state abstraction predicates and abstract actions. State and action abstractions compactly represent environment dynamics and robot capabilities. For `cleanhouse`, state abstraction defines predicates over spatial relations between objects, such as `frontIsClear()` or `present(n)` where `n` is an object of interest, to capture task-relevant context. Action abstraction encodes low-level robot skills—either primitive or learned—that serve as modular building blocks, like `move()`, `turnLeft()`, or `pickUp(n)`. Together, these abstractions enable generalizable, high-level decision-making policies.

```
def P (n: can):
  while (¬present(n)):
    while (¬present(n)):
      if (leftIsClear()):
        turnLeft()
        while(leftIsClear()):
          move()
      if (¬frontIsClear()):
        turnRight()
    move()
  pickUp(n)
```

Figure 3. Robot control program synthesis for the `cleanhouse` benchmark.

The main idea of ReGuS in addressing the exploration challenge in RL is the use of loops. In multi-object environments, encapsulating the learned procedure for handling one object within a loop allows the agent to efficiently generalize this behavior across all similar objects, resulting in more structured exploration. ReGuS supports hierarchical loop synthesis to efficiently reduce the search space for loop-based programs. At the high level, ReGuS synthesizes a loop sketch that captures the skeletal structure of a program with undetermined loop bodies. The low-level sketch completion phase then provides feedback, guiding the high-level synthesizer. This feedback, given as the highest reward achieved by the low-level synthesizer to fill out a loop sketch, helps prioritize the search path for the best quality loop sketch. For example, for `cleanhouse`, ReGuS might discover a loop sketch: **while** (`not present(can)`) { $??_{C1}$ }; $??_{C2}$; that can lead to a program that picks up at least one trash can. The high-level synthesizer prioritizes sketches like this and may expand it as **while** (`not present(can)`) {**while** (`not present(can)`) { $??_{C1}$ }; $??_{C2}$}; $??_{C3}$; for deriving a program that iteratively retrieves all trash cans, thus significantly expediting the environment exploration process. ReGuS also supports on-demand synthesis of conditional statements and curriculum-guided procedure synthesis for continual learning. These strategies make ReGuS far more data-efficient than existing learning and synthesis approaches to overcome the exponential growth of the program search space. For example, ReGuS solves `cleanhouse` in 1.6M environment steps—orders of magnitude fewer than the 20M required by a stochastic synthesizer (Carvalho et al., 2024), while LLM-based methods e.g. (Liang et al., 2023) fail to scale due to the difficulty of generating the complex control flow as required in Fig. 3.

2. Symbol Discovery via Abstraction Refinement for Neurosymbolic Programs

A crucial bottleneck in existing synthesis techniques for RL and robotics systems, including ours π-PRL and ReGuS (described in Sec. 1), is the reliance on manually designed symbolic definitions of state abstraction predicates and abstract actions in a DSL. High-quality abstractions often require significant human effort and domain knowledge to customize effectively. My work explores ways to automatically generate such abstraction as part of the synthesis process.

2.1 Comparative Abstraction Refinement. Automatically learning state and action abstractions has been a key area in RL and robotics. While prior work has explored learning relational abstractions from task demonstrations, existing approaches either assume access to low-level controllers that are already available or predefined predicates—limiting their applicability.

The simultaneous discovery of both remains an open challenge. My research asks: *can we develop abstraction refinement techniques to automatically infer suitable state and action abstractions from task demonstrations?*.

We introduced RoboScribe (Cui et al., 2025), a synthesis engine based on Comparative Abstraction Refinement (Fig. 4). RoboScribe identifies critical subgoals by contrasting successful demonstrations with failed executions of synthesized programs. It learns *state predicates* that define such subgoals and *abstract actions* grounded as low-level policies that transition between them, recovering the task's hierarchical structure. RoboScribe starts with a coarse abstraction `True` $\rightsquigarrow \psi_R$, assuming a single (neural) policy can transition any initial state to the goal condition ψ_R which can be accessed via a black-box reward. If no single policy can solve the task under this abstraction, RoboScribe refines it by learning predicates for key intermediate states. For example, in `pick&place` (Fig. 4), the initial abstraction recognizes only goal states where the block is placed at the red target zone (ψ_R). However, a controller trained to directly satisfy ψ_R often fails to realize that it has to grasp the block first as exemplified in Fig. 4. RoboScribe refines the abstraction by comparing states from failure trajectories with those preceding success in demonstrations, learning a predicate φ that captures a key subgoal: the gripper positioned above the block: `True` $\rightsquigarrow \varphi \rightsquigarrow \psi_R$. By recursively refining abstractions in this way, RoboScribe decomposes complex tasks into subtasks (e.g. `True` $\rightsquigarrow \varphi$ and $\varphi \rightsquigarrow \psi_R$), enabling efficient learning of abstract states and actions.

In addition to symbol discovery for NSP, RoboScribe serves as an efficient neurosymbolic synthesis algorithm in its own right. The key feature is the use of on-the-fly learned state abstraction predicates to identify repeating subroutines from demonstrations, enabling synthesis of state-conditioned loops that generalize to tasks with unbounded objects. For example, consider the `Tower` task of manipulating a robot arm to stack blocks into a tower in Fig. 1a. Predicates for subgoals like grasping and lifting a block recur across different blocks throughout the demonstration. RoboScribe first constructs a loop program skeleton with its loop body as a sequence of abstract actions to achieve the recurring subgoals, then fills in an object-ordering predicate as the loop guard to determine the object processing order, and finally optimizes action abstractions (as low-level neural policies) to ensure robust generalization across all loop iterations. In tasks like `Tower`, the object-ordering predicate is critical. RoboScribe infers ordering predicates from demonstrations to capture task intent—why certain objects are handled before others. Experimental results show that RoboScribe generalizes to long-horizon, multi-object tasks, outperforming deep RL and programmatic RL baselines in interpretability and efficiency—for example, programs for stacking towers transfer to pyramids without retraining.

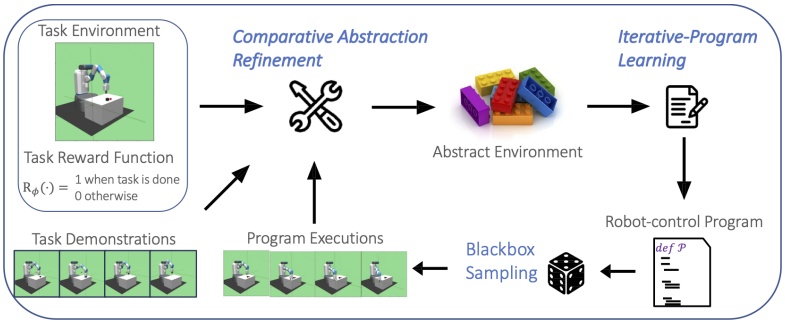


Figure 4. Overview of the RoboScribe framework.

2.2 Learning from Exploration: Bootstrapping Demonstrations for Synthesis. A major limitation in RoboScribe is the need for manual task demonstration provision (Fig. 4) for comparative abstraction refinement. My research explores: *can environment abstraction be fully automated by allowing an agent to explore and collect successful trajectories*? When tasks are challenging, exploration alone may only yield agents with low to medium success rates. However, by combining this with comparative abstraction refinement, the agent can actively learn from its own experience—identifying key differences between successful and failed trajectories to better reason about how to complete the task. Environment exploration remains a key challenge in the RL literature, as it is often difficult to determine which actions ultimately lead to goal-satisfying behavior. I proposed CE^2 (Duan et al., 2024a), "Cluster Edge Exploration", a new goal-directed exploration algorithm. In this context, a "goal" is a state in the robot environment. The key idea is clustering to group states that are easily reachable from one another by the current policy under training in a latent space. By traversing to states on the boundary of these clusters before doing exploratory behavior, CE^2 enables the agent to progressively broaden the coverage of each state cluster to effectively explore a novel environment. Building upon CE^2, we also developed MUN (Duan et al., 2024b) and Cago (Duan et al., 2025) that further improve the agent's exploration ability. Integrating these environment exploration mechanisms with RoboScribe, as shown in Fig. 2, leads to a fully automatic robot-control program synthesis framework.

3. Neurosymbolic Program Verification

A central thrust of my research lies in integrating learning and verification to ensure that neurosymbolic programs satisfy rigorous correctness specifications. My core algorithmic insight is to exploit the hierarchical structure of neurosymbolic

programs to enable (i) deductive verification of high-level symbolic programs by abstracting neural components in abstract state spaces, and (ii) modular verification of low-level neural components within the concrete state spaces. This separation allows for efficient verification of the program's logical structure without being hindered by the complexity of its neural components. The neural components are then compositionally verified within their concrete domains.

3.1 High-level Deductive Verification. Neurosymbolic programs synthesized by ReGuS (Sec. 1) or RoboScribe (Sec. 2) often incorporate unbounded loops to enable generalization across environments with varying numbers of objects. Since these programs are learned from finite demonstrations, a key challenge is to formally verify that they can scale to arbitrary object counts. My research asks: *how do we analyze the interaction between high-level programs and low-level neural components to ensure the logical correctness of the overall control strategy?*

Our key idea is to perform high-level program verification entirely within abstract state and action spaces by leveraging the specifications of neural components. These specifications—expressed as pre- and post-conditions—are already inferred through RoboScribe's comparative environment abstraction refinement. For example, Fig. 5 depicts a synthesized program STACK by RoboScribe for the `Tower` task in Fig. 1a for stacking blocks on the table to a base block b_0. The program iteratively extracts a block b' such that no other block is above it (i.e., $\texttt{top}(b')$ is true), and b' is not the top block b of the current partial stack. It then uses a neural control policy π to move b' onto block b. The neural policy has a postcondition learned by RoboScribe as $\texttt{on}(b', b)$. A loop invariant (Inv in Fig. 5) to verify the high-level program ensures that each block either remains on the table or has already been placed in the current partial stack above b_0. In this example, the task specification and invariant are universally quantified formulas over relations. They fit in the decidable Bernays-Schonfinkel fragment (we use additional axioms to constrain the transitive closure $\texttt{on}^*$ over the `on` predicate). My work studies loop invariant inference from demonstrations and uses SMT-based verification to discharge the resulting verification conditions.

procedure STACK(b_0)
  Requires: $\forall\alpha.\ \texttt{top}(\alpha)$
  Ensures: $\forall\alpha.\ \alpha\langle\texttt{on}^*\rangle b_0$
  $b \leftarrow b_0$
  while $\exists b'.\ \texttt{top}(b') \wedge b' \neq b$ **do**
    Inv: $(\forall\alpha.\ \alpha\langle\texttt{on}^*\rangle b_0 \vee \texttt{top}(\alpha))$
      $\wedge\ b\langle\texttt{on}^*\rangle b_0 \wedge \texttt{top}(b)$
    $\pi(b', b)$ $[\texttt{on}(b', b)]$
    $b \leftarrow b'$

Figure 5. Synthesized program by RoboScribe for Tower (Fig. 1a).

3.2 Proof Space Optimization. Once high-level neurosymbolic programs are verified in the abstract space, we must ensure that each neural component within satisfies its specification—expressed as pre- and post-conditions (e.g., those inferred by RoboScribe). For example, we verify that the neural policy $\pi(b', b)$ in Fig. 5 can reliably stack b' `on` b. My research investigates: *How do we verify control policies are downward executable in the low-level continuous state space?*

Rather than adopting a post-hoc verification approach—where a model is trained without regard to its verifiability and only checked afterward—I develop learning frameworks that incorporate verification directly into the training loop. Our research studies how a learner can interact with a verifier during synthesis to guide the learning process toward solutions that are not only effective but also provably correct with respect to formal constraints. We developed ART (Lin et al., 2020) and VEL (Wang & Zhu, 2023), a verification-guided framework that integrates formal abstraction-based verification with π-PRL (Sec. 1) to synthesize control policies for continuous-state robot environments. VEL iteratively refines a candidate controller using verifier feedback, treating the controller and environment as a closed-loop system. A key challenge is handling spurious counterexamples caused by overapproximations in reachability analysis (e.g., using Taylor models (Chen et al., 2013)), which lead to error accumulation over time (the wrapping effect). VEL addresses this by lifting π-PRL's synthesis from training examples to the abstraction space, enabling optimization against both real and spurious violations.

3.3 Safe Learning. A potential limitation of VEL is its reliance on a known environment model (which is often difficult to obtain accurately). My research explores: *In settings with unknown environment dynamics, how can we reduce safety risks and irreversible damage caused by the robot during exploratory training?*

We introduced VELM (Wang & Zhu, 2024), a model-based RL framework grounded in verification principles to ensure safe exploration in unknown environments, enabling agents to learn while consistently satisfying safety constraints. The main idea of VELM is to learn environment models as symbolic formulas and conduct formal reachability analysis over the learned models for safety verification with probabilistic guarantees. Compared to neural models learned by existing work, symbolic environment models are conducive to long-horizon reachability analysis, uniquely enabling efficient computation of the reachable set for a control system. An online shielding layer is then constructed to confine the RL agent's exploration solely within a state space verified as safe in the learned model, thereby bolstering the overall safety profile of the RL system. Across diverse RL environments, we showed that VELM significantly reduces safety violations compared to existing safe learning techniques, all without compromising the RL agent's reward performance.

4. Other Works

In addition to neurosymbolic programming, I have pursued complementary directions that share a unifying goal: enabling learning systems to reason with and be constrained by formal structures. These efforts explore how formal logic—particularly Linear Temporal Logic (LTL)—can be used not only to specify complex objectives for reinforcement learning agents, but also to guide exploration, generalization, and verification. They also extend to data-driven methods for formal verification, aiming to make correctness guarantees more scalable and accessible by learning abstractions and invariants. Together, these lines of work contribute to a broader vision of safe, verifiable, and generalizable programming systems.

4.1. Temporal Logic Learning and Verification

Temporal Logic Learning. LTL has been extensively studied as an alternative framework for specifying RL objectives, but ensuring their satisfaction over infinite horizons is challenging—especially in high-dimensional, continuous systems where sample efficiency is critical. I have developed two complementary approaches to LTL-constrained control policy optimization. The first, GCRL-LTL (Qiu et al., 2023), shows that goal-conditioned RL agents, when properly guided, can generalize zero-shot to arbitrary LTL specifications without additional training over the LTL task space. The second work, TiLoIL (Fan & Zhu, 2025), introduces an imitation learning method for LTL objectives. The key idea is to leverages Limit Deterministic Büchi Automata (LDBA) for LTL objectives and encourage agents to reach LDBA-accepting states repeatedly and reliably. To achieve this, TiLoIL segments both expert demonstrations and agent trajectories based on visits to LDBA-accepting states. It then trains the agent to imitate expert behavior within each segment, enabling it to reach these accepting states from arbitrary positions. This segmentation strategy improves sample efficiency and helps agents satisfy complex LTL specifications without requiring long and dense demonstrations.

Temporal Logic Verification. Existing verification efforts for RL controllers primarily focus on goal-reaching tasks, leaving the verification of richer temporal logic specifications largely unaddressed. My work presents a deductive synthesis framework, VEL-∞ (Wang & Zhu, 2025), that ensures RL policies satisfy temporal logic specifications over infinite horizons. By decomposing temporal objectives into subtasks, our method efficiently synthesizes and verifies RL policies for each subtask based on VEL (Wang & Zhu, 2023). Formal verification guarantees that the postcondition of one subtask serves as the precondition for the next, enabling their cyclic composition. This invariant ensures that the synthesized policies can execute indefinitely while maintaining correctness. Experiment results show that VEL-∞ outperforms standard RL methods for temporal logic objectives in both performance and adherence to safety and liveness constraints.

4.2. Data-Driven Formal Verification

Despite decades of investigation, software systems remain vulnerable to code defects. My research explored data-driven techniques to enable scalable automated verification of software systems. My work has focused on (a) validating complex data structures properties (Zhu et al., 2016), (b) discovering sound invariants of complex control-flow processes (i.e., loops and recursion) (Zhu et al., 2018), and (c) reasoning over new computational models (i.e., machine learning models) with complex non-linear behavior (Zhu et al., 2019). My main approach consists in discovering high-level abstract specifications capturing the programmer's intent (i.e., reachability specifications in the case of data structures (Zhu et al., 2015b; 2016); inductive invariants in the case of loops and recursion (Zhu et al., 2015a; 2018); and, deterministic programs in the case of machine learning models (Zhu et al., 2019; Lin et al., 2020)) to enable efficient verification that would be difficult or impossible otherwise. The key technical insight is the application of data-driven techniques to a formal methods toolchain to guide the automated discovery of high-level abstract specifications from code.

5. Grand Vision of Future Work

My future work will continue to unify ML with PL and FM through neurosymbolic synthesis for developing learning-enabled systems that are both verifiable and interpretable. I envision a future where learning-enabled systems move beyond end-to-end black-box learning toward the synthesis of structured, interpretable, and verifiable programs that retain the adaptability of learning. In this paradigm, agents will autonomously acquire reusable, verifiable skills that can be composed into complex behaviors through program synthesis, guided by formal specifications. Verification will become a continual process—not an afterthought—interleaved with learning to ensure correctness and safety in open-ended environments. This tight integration of synthesis and verification with learning will be key to building reliable, general-purpose autonomous systems that can reason, adapt, and act safely in unstructured settings such as homes, factories, and disaster zones.

Looking ahead, I see Large Language Models (LLMs) as semantic priors for abstraction learning and program synthesis of interpretable learning-enabled systems. By leveraging their generative capabilities, LLMs can propose symbolic program scaffolds grounded in high-level task descriptions or goal specifications. However, current methods often rely on fixed state abstractions or hardcoded skill libraries for LLM programming, limiting its flexibility and generalization to novel tasks. Integrating LLMs with RoboScribe within a hybrid architecture is a promising direction: LLM-generated structures—though potentially noisy or incomplete—can initiate the synthesis process, which is then refined by learning low-level behaviors and guided by verification-driven abstraction refinement to ensure safe and correct execution. My ongoing work (Mao et al., 2025) provides preliminary evidence supporting the feasibility of this approach.

Another central thrust of my future research vision is to enable robot-control program synthesis directly from large-scale offline datasets, reducing the reliance on costly robot-environment interaction. Leveraging large-scale offline data has recently emerged as a powerful trend in robot learning, offering a promising alternative to expensive and risky real-world exploration. Building on RoboScribe, a promising direction is to contrast high-performing and medium- or low-performing trajectories within offline datasets to automatically discover useful abstractions that capture the essential structure of tasks. In parallel, these datasets can be used to learn accurate world models that can be used as stand-ins for real environments during neurosymbolic program synthesis. A key theoretical goal is to bound the performance gap between synthesis in learned models and online interaction, providing principled assurances for deploying such systems. As systems transition from offline to online learning, robots will gain the ability to interact with formal abstractions and incrementally refine their understanding of both task structure and environment dynamics, leading to more adaptive and verifiable behaviors over time. My ongoing work (Qiu et al., 2025) explores the viability of this idea.

I plan to extend my work on neurosymbolic robot programming to other NSP domains that require structured reasoning. For example, in mathematical theorem proving, while most existing approaches learn a prover at the tactic level—mapping proof states to individual tactics—I propose using comparative abstraction refinement to synthesize strategy-level proof systems. In this setting, a strategy corresponds to a sequence of tactics, restructured as a lemma. A meta-synthesizer can then learn to predict which strategies to invoke, composing new ones as needed during the proof search. This approach has the potential to improve generalization, modularity, and interpretability in a wide range of automated reasoning systems.

I envision extending the core principles of specification-satisfying trajectory exploration and comparative abstraction refinement to ordinary software systems. Modern software systems are increasingly complex and dynamic, making it difficult to manually design abstractions or verify correctness at scale. By applying comparative abstraction refinement, we could automatically discover critical structural patterns by contrasting successful and erroneous executions, uncovering abstractions that guide both synthesis and verification. Such an approach would naturally enable compositional reasoning by separately reasoning about components and assembling their guarantees. This line of work opens the door to a future in which software systems can learn, adapt, and verify themselves.

References

Carvalho, T. H., Tjhia, K., and Lelis, L. Reclaiming the source of programmatic policies: Programmatic versus latent spaces. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Chaudhuri, S., Ellis, K., Polozov, O., Singh, R., Solar-Lezama, A., and Yue, Y. Neurosymbolic programming. *Found. Trends Program. Lang.*, 7(3):158–243, 2021.

Chen, H., Li, Y., Shi, S., Liu, S., Zhu, H., and Zhang, Y. Graph collaborative reasoning. In *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining*, pp. 75–84. ACM, 2022a. URL https://doi.org/10.1145/3488560.3498410.

Chen, H., Li, Y., Zhu, H., and Zhang, Y. Learn basic skills and reuse: Modularized adaptive neural architecture search (MANAS). In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 169–179. ACM, 2022b. URL https://doi.org/10.1145/3511808.3557385.

Chen, X., Ábrahám, E., and Sankaranarayanan, S. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification - 25th International Conference, CAV*, volume 8044 of *Lecture Notes in Computer Science*, pp. 258–263. Springer, 2013.

Cui, G. and Zhu, H. Differentiable synthesis of program architectures. In *Advances in Neural Information Processing Systems*

34: Annual Conference on Neural Information Processing Systems (NeurIPS), 2021. URL https://openreview.net/pdf?id=ivXd1iOKx9M.

Cui, G., Wang, Y., Qiu, W., and Zhu, H. Reward-guided synthesis of intelligent agents with control structures. *Proc. ACM Program. Lang.*, 8(PLDI):1730–1754, 2024. URL https://herowanzhu.github.io/PLDI_2024.pdf.

Cui, G., Wang, Y., Mao, W., Duan, Y., and Zhu, H. Abstraction refinement-guided program synthesis for robot learning from demonstrations. https://herowanzhu.github.io/roboscribe.pdf, 2025. Under review at the OOPSLA issue of the Proceedings of the ACM on Programming Languages (PACMPL), 2025. Initial reviews: 5 (Accept), 4 (Weak Accept), 6 (Clear Accept).

Duan, Y., Cui, G., and Zhu, H. Exploring the edges of latent state clusters for goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024a. URL https://openreview.net/pdf?id=9hKN99RNdR.

Duan, Y., Mao, W., and Zhu, H. Learning world models for unconstrained goal navigation. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024b. URL https://openreview.net/pdf?id=aYqTwcDlCG.

Duan, Y., Wang, Y., Qiu, W., and Zhu, H. Learning from demonstrations via capability-aware goal sampling. https://herowanzhu.github.io/Cago.pdf, 2025. Under review at NeurIPS 2025.

Fan, Z. and Zhu, H. Imitation learning with temporal logic constraints. https://herowanzhu.github.io/TiLoIL.pdf, 2025. Under review at NeurIPS 2025.

Inala, J. P., Bastani, O., Tavares, Z., and Solar-Lezama, A. Synthesizing programmatic policies that inductively generalize. In *8th International Conference on Learning Representations, ICLR*, 2020.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation, ICRA*, 2023.

Lin, X., Zhu, H., Samanta, R., and Jagannathan, S. Art: Abstraction refinement-guided training for provably correct neural networks. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020*, pp. 148–157. IEEE, 2020. URL https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_22.

Mao, W., Duan, Y., Qiu, W., and Zhu, H. Compositional policy optimization with language models. https://herowanzhu.github.io/L2S.pdf, 2025. Under review at NeurIPS 2025.

Patton, N., Rahmani, K., Missula, M., Biswas, J., and Dillig, I. Programming-by-demonstration for long-horizon robot tasks. *Proc. ACM Program. Lang.*, 8(POPL):512–545, 2024.

Qiu, W. and Zhu, H. Programmatic reinforcement learning without oracles. In *The Tenth International Conference on Learning Representations, ICLR*. OpenReview.net, 2022. URL https://openreview.net/forum?id=6Tk2noBdvxt.

Qiu, W., Mao, W., and Zhu, H. Instructing goal-conditioned reinforcement learning agents with temporal logic objectives. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems NeurIPS*, 2023. URL https://openreview.net/pdf?id=19AgWnmyoV.

Qiu, W., Cui, G., Duan, Y., and Zhu, H. Learning from sparse-reward offline datasets via preference-based policy optimization. https://herowanzhu.github.io/PREFORL.pdf, 2025. Under review at NeurIPS 2025.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, 2018.

Wang, Y. and Zhu, H. Verification-guided programmatic controller synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS*, volume 13994 of *Lecture Notes in Computer Science*, pp. 229–250. Springer, 2023. URL https://herowanzhu.github.io/vel_tacas.pdf.

Wang, Y. and Zhu, H. Safe exploration in reinforcement learning by reachability analysis over learned models. In *Computer Aided Verification - 36th International Conference, CAV*, volume 14683 of *Lecture Notes in Computer Science*, pp. 232–255. Springer, 2024. URL https://herowanzhu.github.io/CAV_2024.pdf.

Wang, Y. and Zhu, H. Deductive synthesis of reinforcement learning agents for infinite horizon tasks. https://herowanzhu.github.io/VEL-inf.pdf, 2025. Computer Aided Verification - 37th International Conference, CAV.

Zhu, H., Nori, A. V., and Jagannathan, S. Learning refinement types. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP*, pp. 400–411. ACM, 2015a. URL https://doi.org/10.1145/2784731.2784766.

Zhu, H., Petri, G., and Jagannathan, S. Poling: SMT aided linearizability proofs. In *Computer Aided Verification - 27th International Conference, CAV*, volume 9207 of *Lecture Notes in Computer Science*, pp. 3–19. Springer, 2015b. URL https://doi.org/10.1007/978-3-319-21668-3_1.

Zhu, H., Petri, G., and Jagannathan, S. Automatically learning shape specifications. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI*, pp. 491–507. ACM, 2016. URL https://doi.org/10.1145/2908080.2908125.

Zhu, H., Magill, S., and Jagannathan, S. A data-driven CHC solver. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI*, pp. 707–721. ACM, 2018. URL https://doi.org/10.1145/3192366.3192416.

Zhu, H., Xiong, Z., Magill, S., and Jagannathan, S. An inductive synthesis framework for verifiable reinforcement learning. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI*, pp. 686–701. ACM, 2019. URL https://doi.org/10.1145/3314221.3314638.