Synthesizing Programmatic Policy for Generalization within Task Domain

Tianyi Wu , **Liwei Shen** , **Zhen Dong*** , **Xin Peng** and **Wenyun Zhao**
Fudan University
{tywu18, shenliwei, zhendong, pengxin, wyzhao}@fudan.edu.cn

Abstract

Deep reinforcement learning struggles to generalize across tasks that remain unseen during training. Consider a neural process observed in humans and animals, where they not only learn new solutions but also deduce shared subroutines. These subroutines can be applied to tasks involving similar states to improve efficiency. Inspired by this phenomenon, we consider synthesizing a programmatic policy characterized by a conditional branch structure, which is capable of capturing subroutines and state patterns. This enables the learned policy to generalize to unseen tasks.The architecture of the programmatic policy is synthesized based on a context-free grammar. Such a grammar supports a nested If-Then-Else derivation and the incorporation of Recurrent Neural Network. The programmatic policy is trained across tasks in a domain through a meta-learning algorithm. We evaluate our approach in benchmarks, adapted from PDDLGym for task planning and Pybullet for robotic manipulation. Experimental results showcase the effectiveness of our approach across diverse benchmarks. Moreover, the learned policy demonstrates the ability to generalize to tasks that were not seen during training.

1 Introduction

Deep reinforcement learning has made significant breakthroughs across many control tasks. However, the ability to generalize across diverse tasks in a task domain remains a challenge, even for state-of-the-art deep reinforcement learning algorithms [Packer *et al.*, 2018]. Specifically, consider the 'Tower of Hanoi' task, characterized by three pillars and a set of discs of varying sizes. The primary objective is to transfer all the discs from the source pillar to the target pillar while adhering to specific constraints. The variations of the task occur when the number of discs changes; for example, a control policy learned from a 'Tower of Hanoi' task with three discs. When evaluating the policy in the 'Tower of Hanoi' with four discs, it exhibits poor performance under these conditions. The primary distinction in the 'Tower of Hanoi' task domain is related to the number of discs. Through summarizing the similar states across the tasks, it is found that the 'Tower of Hanoi' with different numbers of discs shares the same substructure. It is feasible to obtain a general solution applicable across tasks based on identifying state patterns and reusing subroutines. Thus, we focus on the problem of learning such a policy capable of generalizing in a task domain with different variations such as differences in number or position, where the underlying dynamics are the same.

When examining a neural process in humans and animals, they possess the capability not only to acquire novel solutions but also to deduce subroutines and apply them to similar tasks. These subroutines can be applied to similar tasks to improve efficiency. Inspired by this phenomenon, we consider synthesizing a programmatic policy characterized by a conditional branch structure, which is capable of capturing shared subroutines and state patterns across different tasks in one domain. Such a policy consists of a set of conditional branch structures. Each condition is designed to capture the state patterns. Each branch is applied to computes an action as return for the agent to execute, expected to deduce a subroutine. By doing so, the programmatic policy is expressive to capture tasks of interest, e.g., subroutines used to improve decision making and state patterns facilitating the selection of subroutines. This enables the learned policy to generalize to unseen tasks, which involves shared subroutines and similar states.

Algorithms have been proposed to learn programmatic policies that generalize better than traditional neural network policies [Verma *et al.*, 2018; Verma *et al.*, 2019], while necessitating user demonstrations. A programmatic state machine policy [Inala *et al.*, 2020] that inductively generalizes to the tasks requiring repetition. A programmatic policy with a conditional branch structure is proposed in [Qiu and Zhu, 2022], but the generalization across different tasks is not evaluated. The architecture of the programmatic policy is synthesized based on a context-free grammar, which is a tailored domain-specific language. The DSL describes the definition of a nested If-Then-Else structure. We also incorporate Recurrent Neural Network (RNN) blocks to synthesize the architecture of the programmatic policy. The hidden states of the RNN are retained across different episodes throughout the agent-environment interactions. This enables the agent to re-

*Corresponding author.

member past knowledge and apply it to new tasks. A task distribution can be obtained by varying the certain aspects of the environment settings from a task domain. We utilize a meta-learning algorithm to train the programmatic policy across the task distribution.

We benchmark our method against the state-of-the-art reinforcement learning baseline methods and ablation methods in seven benchmarks, which are adapted from PDDLGym for task planning and Pybullet for robotic manipulation. The results demonstrate that the programmatic policy trained via our method exhibits generalization across different tasks. It also performs well in the more complex task that have not been encountered before.

Our method draws inspiration from a real-world scenario involving real robot tasks. We operate under the assumption that the value of the variations within an environment are unlikely to exhibit large fluctuation. For instance, consider the 'Tower of Hanoi' environment, where complexity escalates exponentially as the number of discs increases. Completing such a task with thousands of operations becomes nearly impossible for a robot, resulting from the risk of operational failure in real-world environment. Therefore, the the values of the variations we consider are all adhere to a reasonable range in this paper.

2 Method Overview

2.1 Problem Formalization

We consider a task distribution denoted as $p(\mathcal{T}_H)$, in which H represents a collection of tasks. An agent with a programmatic policy is in the form of $\mathcal{P}_{E,\theta}$, where the θ presents the parameter of the policy. The parameter E denotes the program architecture which can be defined by Domain Specific Language (DSL). S and A represent the states and actions across the tasks. The problem is to train a policy $\mathcal{P}_{E,\theta}(a_t \mid s_t)$, that can generalize within $\mathcal{T}_H$. Formally, we model each task $\tau_i \in \mathcal{T}_H$ as a Markov Decision Process defined by a tuple $\{S_i, A_i, T_i, R_i\}$ where S_i and A_i denote the environmental observation and action spaces, respectively. Furthermore, $T_i : S_i \times A_i \times S_i \to [0, 1]$ represents state transition probabilities, and $R_i : S_i \times A_i \to \mathbb{R}$ quantifies the corresponding rewards when transitioning between states. The parameter of the policy $\widehat{\theta}_i$ is estimated by maximizing the cumulative discounted reward $E_{s_0,a_0,s_1\cdots\sim\mathcal{P}_{E,\theta_i}}[\sum_0^\infty \gamma^t \cdot R_i(s_t, a_t)]$ where $\gamma \in (0, 1]$. Subsequently, we update the policy $\mathcal{P}_{E,\theta}$ by the θ_i learned from each task.

2.2 Architecture Synthesis

A programmatic policy processes an environmental state as input and produces an action for the agent. Drawing inspiration from the program architecture search framework [Qiu and Zhu, 2022], we infer the architecture of a programmatic policy denoted as $\mathcal{P}_{E,\theta}$ by DSL.

The DSL is represented in Backus-Naur form [Winskel, 1993]. A context-free grammar, depicted in Figure 1, is crafted to define the programs to be learned. The E is nonterminals, representing the program. The C is an affine transformation to compute an action value. B is evaluated by RNN layers which acts as a condition. This design facilitates

$$E ::= C \mid \text{if } B \text{ then } C \text{ else } E$$
$$B ::= RNN\_Layer(\mathcal{X}) \geq 0$$
$$C ::= \theta_c + \theta^{\mathrm{T}} \cdot \mathcal{X}$$

Figure 1: DSL for programmatic policy

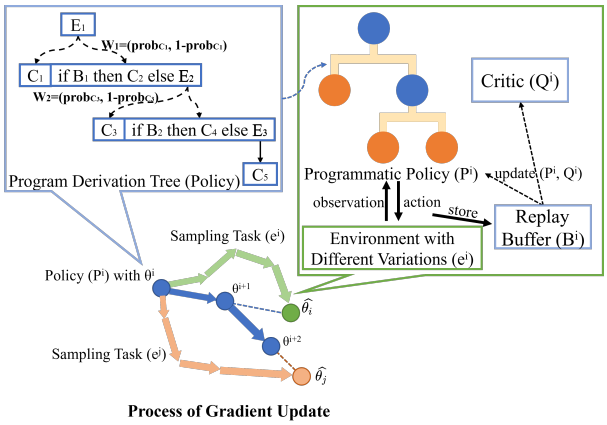


Figure 2: Illustration of training programmatic policy

the retention of knowledge across various tasks. $\mathcal{X} \in \mathbb{R}^m$ represents the state observation, m denoting the dimension. This DSL allows the derivation of programs. As an example, we can deduce a program as $if\ B_1\ then\ C_1\ else\ E_1$ to $if\ B_1\ then\ C_1\ else\ (if\ B_2\ then\ C_2\ else\ E_2)$. The semantics of the program, such as $if\ B_1\ then\ C_1\ else\ E_1$, are computed by a function denoted as $[\![if\ B_1\ then\ C_1\ else\ E_1]\!](x)$, where the variable x serves as input to the **If-Then-Else** program. yielding a real-valued vector as output. $C ::= \theta_c + \theta^{\mathrm{T}} \cdot \mathcal{X}$, yielding a real-valued vector as output, is an affine transformation, where $\theta \in \mathbb{R}^{m\times n}$ and n represents the dimension of the action spaces.

To ensure differentiability, the program derived from the DSL can be interpreted as a numerical approximation:

$$[\![if\ B\ then\ C\ else\ E]\!](s)$$
$$= \sigma([\![B]\!](s)) \cdot [\![C]\!](s) + (1 - \sigma([\![B]\!](s))) \cdot [\![E]\!](s)$$

, where σ represents the sigmoid function. By utilizing the sigmoid function, the **If-Then-Else** program is transformed into a differentiable expression with binary branch structure. The output of the sigmoid function represents the probability of selecting a particular branch. The definition of the policy architecture aids to capture the subroutines and state patterns.

2.3 Training of Programmatic Policy

The left part of Figure 2 illustrates a program derivation tree with a depth of three. The blue circle represents E which can be expanded to a **if-then-else** branch, and orange represents C which is to output the action value. According to the DSL expression, we can systematically expand a program into a program derivation tree. The program derivation tree represents all possible program derivations within a specified depth limit for program abstract syntax trees. From Figure 2, $\mathcal{W}$ is a vector containing the knowledge of the selection of

Algorithm 1: Algorithm for training programmatic policy

Input: Distribution over tasks $p(\mathcal{T}_H)$, Learning rate α, Meta Learning rate β, DSL E, Depth d
Output: Trained policy $\mathcal{P}_\theta$

1 Derive Programmatic Policy $\mathcal{P}_\theta$ via E and d
2 Initialize $\theta = \{\mathcal{W}, \varphi\}$
3 **while** *not done* **do**
4 Sample task $\mathcal{T}_i \sim p(\mathcal{T}_H)$
5 Sample Trajectories with $\mathcal{P}_\theta$
6 Store in ReplayBuffer B_i
7 Using PPO to estimate $\widehat{\theta}_i$ with learning rate α
8 $\theta_{i+1} \leftarrow \theta_i + \beta(\theta_i - \widehat{\theta}_i)$
9 **end**
10 Extract $\mathcal{P}_{\theta=\{\varphi\}}$ by fixing an optimal $\mathcal{W}$
11 **while** *not done* **do**
12 Sample task $\mathcal{T}_j \sim p(\mathcal{T}_H)$
13 Sample Trajectories with $\mathcal{P}_\theta$
14 Store in ReplayBuffer B_j
15 Using PPO to estimate $\widehat{\theta}_j$ with learning rate α
16 $\theta_{j+1} \leftarrow \theta_j + \beta(\theta_j - \widehat{\theta}_j)$
17 **end**

each layer's architecture. Each digit in $\mathcal{W}$ models a binary selection. For instance, $prob_{C_1}$ represents the probability of expanding E_1 into C_1 and $1 - prob_{C_1}$ signifies the probability of expanding it into $if\ B_1\ then\ C_2\ else\ E_2$. The value of $prob_{C_1}$ is calculated using a Softmax function. Besides $\mathcal{W}$, φ is another learnable parameter of the programmatic policy. According to the definition of DSL, φ encompasses the parameters from both the RNN layer denoted as B and the linear layer denoted as C. Thus, the parameter θ can be viewed as a combination of $\mathcal{W}$ and φ, denoted as $\theta = \{\mathcal{W}, \varphi\}$. As E derived from a DSL, E can be regarded as a constant. Then we focus on optimizing the θ of policy $\mathcal{P}$.

In this paper, we consider Proximal Policy Optimization [Schulman *et al.*, 2017] as the foundational reinforcement learning algorithm to train the policy as depicted in the right part of Figure 2. The actor P^i represents the programmatic policy we defined, while the critic Q^i is actually a neural network. We utilize a ReplayBuffer B^i for storing trajectories during training.

We adopt Reptile [Nichol and Schulman, 2018] as the learning algorithm for gradient update, which is shown in the bottom part of Figure 2. It operates by performing a stochastic gradient descent on the sampled tasks and updating the initial parameters toward achieving the final learned parameters specific to the given task. It is worth noting that we omit the fine-tuning process. The algorithm solely needs a black-box optimizer such as SGD or Adam and offers good computational efficiency and performance.

We propose an algorithm to demonstrate the synthesis of the programmatic policy, as in Algorithm 1. The algorithm takes as input training hyperparameters, a DSL description, the maximum depth of the program derivation tree denoted as d, and a task distribution. In line 1, a programmatic policy $\mathcal{P}_\theta$ is deduced in the form of a program derivation tree with depths ranging from 1 to d based on the input DSL. In line 2, we initialize the parameters $\theta = \{\mathcal{W}, \varphi\}$ to be learned. Specifically, we initialize $\mathcal{W}$ with a 50% probability for each branch selection. At the training stage, in line 4, a task is sampled from the task distribution. The trajectories are obtained by agent-environment interactions, in which we design a mechanism to help the agent utilize the RNN blocks. It is elaborated upon in the following section. Subsequently, the trajectories derived from these interactions are stored in a ReplayBuffer for training. PPO is employed to optimize the parameters. We update both $\{\mathcal{W}, \varphi\}$ sequentially using a bilevel optimization technique. According to Reptile, the formula $\beta(\theta_i - \widehat{\theta}_i)$ can be considered as a gradient and subsequently utilized in a more advanced optimizer, such as Adam. After a certain number of training steps, the $\mathcal{W}$ and φ of the policy tend to stabilize. At this point, we can determine the specific architecture of the programmatic policy fixing the $\mathcal{W}$, as depicted in line 10. Specifically, since $\mathcal{W}$ models a series of binary selections, the better architecture of the policy is selected with the maximum likelihood. Since the current policy has only one parameter φ, repeat the above steps to train the policy until it converges, in line 11 to 17.

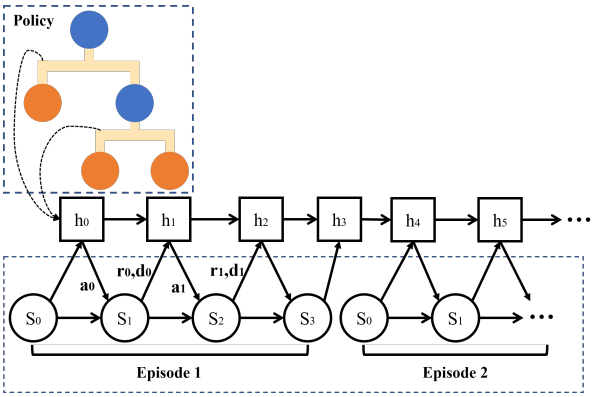


Figure 3: Procedure of agent-environment interaction

2.4 Using RNN to Improve Agent-environment Interactions

The process of the agent interacting with the environment, as described in both line 6 and line 16 of Algorithm 1, is visualized in Figure 3. For each task obtained from a task distribution, multiple episodes are generated through agent's exploration. During each episode, the agent engages with a task environment. Once the agent generates an action a_t, the environment provides the corresponding reward r_t, advances to the next state s_{t+1}, and determines if the episode terminates. This termination status is recorded using the flag d_t, which is set to 1 if the episode ends or left at a default value of 0 otherwise. The input is constructed by combining the following elements: the next state s_{t+1}, action a_t, reward r_t, and termination flag d_t.

As the task changes, the agent must adjust its actions according to the MDP in which it believes it is currently located.

Thus, the agent aggregates all available information on past rewards, actions, and termination flags and continually adapts its policy. To facilitate the agent in learning from prior experience, the programmatic policy incorporates RNN. It utilizes actions and rewards from preceding time steps as training inputs. According to the DSL in Figure 1, certain blocks of the policy comprise recurrent neural network cells. The hidden state h_t is a vector summarized from the programmatic policy. Using the hidden state h_{t+1} and input state s_{t+1} as inputs, the policy generates the subsequent action a_{t+1} and updates the subsequent hidden state h_{t+2}. The policy's hidden state is retained across episodes but is not carried over between distinct tasks.

3 Experiments and Evaluation

We evaluate the effectiveness of our approach on two groups of challenging benchmarks[1], as illustrated in Figure 4. One group comprises three benchmarks—**Hanoi**, **Stacking** and **Hiking** adapted from [Silver and Chitnis, 2020], where the action space is discrete. These benchmarks primarily serve to evaluate our approach in the context of task planning.

- **Hanoi**: Involves various-sized discs stacked in a pyramid formation on the source pillar. The goal is to methodically transfer all the discs, one at a time, to target pillar in Figure 4a. The primary variation in this environment is determined by the number of discs involved. In Hanoi, we encode the observation state as a 1×9 vector.
- **Hiking**: The character is required to traverse pathways and collect all stars on the map. Variations in this environment pertain to the quantity and positioning of these stars, as depicted in Figure 4b. The entire state of the map is encoded as the state of the environment, with the dimension of 1×80.
- **Stacking**: Multiple plates of varying sizes are scattered on a tabletop. The task is to systematically pick up these plates one by one using a gripper and to assemble them in descending order of size, shown in Figure 4c. In this environment, we encode the table status, the stacked plates and the location of the gripper as the environment state, with a dimension of 1×35. Variations in this environment arise from different quantities of plates and their respective positions.

Another group comprises four tasks which are **PandaReach**, **PandaPush**, **PandaSlide**, **PandaStack**. These benchmarks are developed within the Pybullet environment [Gallouédec *et al.*, 2021]. Each environment consists of a Panda robot and a set of manipulable objects. The benchmarks in this group aim to assess our approach in continuous action spaces such as motion control. The variations in the four benchmarks involve differences in both number and position of the manipulable objects, indicated by distinct colors. The target object needs to be manipulated to reach the target point with the corresponding color. When the number and position of the manipulable objects change, each task domain becomes very large.

[1]Code and benchmarks: https://github.com/V0idwu/meta-prl-code

- **PandaReach**: This task involves manipulating the robot's end-effector to reach the target positions, as illustrated in Figure 4d.
- **PandaPush**: The objective is to use the robot's end-effector to push the blocks to the target positions of the corresponding colors, as shown in Figure 4e.
- **PandaSlide**: The goal is to use the robot's end-effector to slide the cylinders to the target positions of the corresponding colors, as depicted in Figure 4f.
- **PandaStack**: The aim of the task is to place the blocks in the correct positions of the corresponding colors and stack them in order, as shown in Figure 4g.

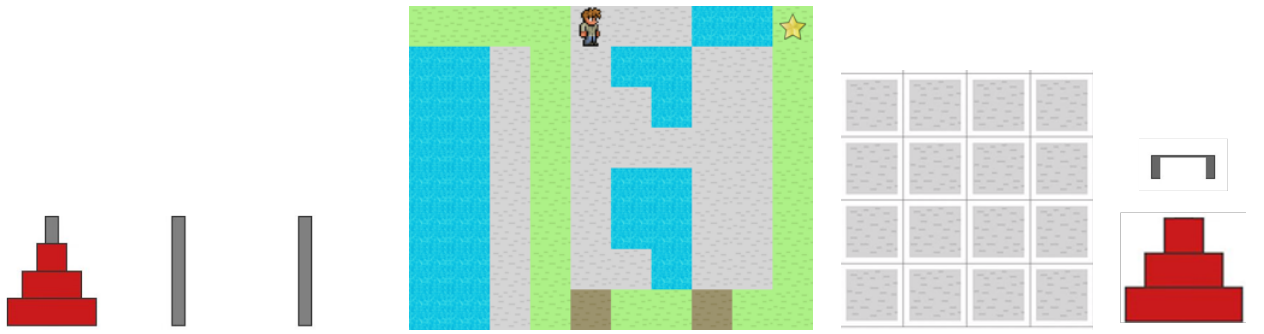
We encode the observation state as a 1×32 vector. The action of the robot is a 4-dimension vector, which contains a 3D coordinate for the end-effector and a finger control.

To highlight the advantages of our approach, we conduct a comparative analysis with the following baselines/ablations:

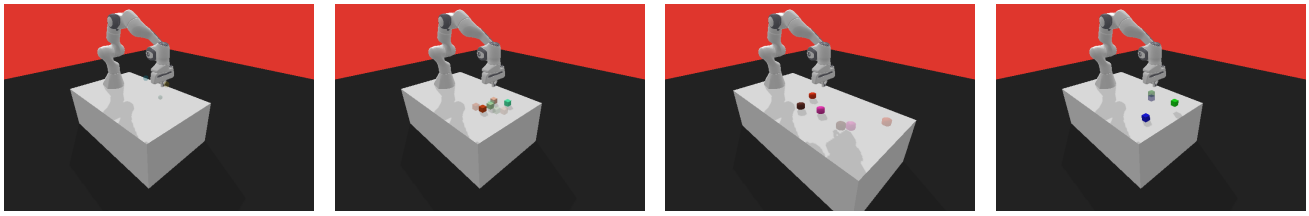
- PPO, SAC: PPO [Schulman *et al.*, 2017], SAC [Haarnoja *et al.*, 2018] serve as baselines in reinforcement learning, trained within a specific environment with fixed variations.
- HIRO: HIRO [Nachum *et al.*, 2018] is a general off-policy algorithm for hierarchical reinforcement learning, trained under identical conditions as PPO.
- PRL: PRL, the Programmatic policy [Qiu and Zhu, 2022], is learned within the framework of PPO and trained under the same conditions.
- ReptilePPO, ReptilePRL: Reptile serves as the foundational meta-learning algorithm. ReptilePPO and ReptilePRL are trained in a task domain with the PPO and PRL respectively.
- PPO-N: Trained within the same meta-learning algorithm of ReptilePPO and ReptilePRL. However, the parameter of the model is not updated in a meta-learning manner.
- Ours: Our approach, based on the ReptilePRL, incorporates an RNN structure and a designed agent-environment interaction.

Performance Table 1 presents an evaluation of the performance of the methods PPO, SAC, HIRO, PRL, ReptilePPO, ReptilePRL, and our approach. We trained the three methods—PPO, SAC, HIRO and PRL in a fixed task, such as Hanoi-3. ReptilePPO, ReptilePRL and our approach are trained across tasks, e.g., the number of the discs in Hanoi ranging from 1 to 4, denoted as Hanoi-1234. Subsequently, we evaluated their performance in the trained tasks and unseen tasks respectively. Evaluation involves measuring the mean and variance of the required action steps for agents to complete tasks. A smaller number of action steps indicates better performance. In the Hanoi benchmark, a single test suffices because the initial state is the same under the different random seeds.

Overall, our method exhibits superior performance in the majority of benchmarks. Moreover, as the environment's complexity increases, the corresponding rise in the required



(a) hanoi (b) hiking (c) stacking



(d) panda-reach (e) panda-push (f) panda-slide (g) panda-stack

Figure 4: Benchmarks for experiments

test envs	PPO	SAC	HIRO	PRL	ReptilePPO	ReptilePRL	Ours
	train envs: Hanoi-3				train envs: Hanoi-1234		
Hanoi-3	187	241	95	423	52	66	**51**
Hanoi-5	290	257	223	493	150	128	**98**
Hanoi-7	386	387	311	500	250	258	**151**
	train envs: Hiking-3				train envs: Hiking-12345		
Hiking-3	210.65 ± 61.66	220.81 ± 80.06	202.00 ± 57.02	117.88 ± 49.32	$\mathbf{21.95 \pm 13.79}$	47.21 ± 18.84	32.18 ± 16.36
Hiking-5	328.44 ± 66.49	268.17 ± 89.64	323.16 ± 61.49	267.31 ± 69.90	129.46 ± 46.49	51.85 ± 24.45	$\mathbf{32.50 \pm 16.10}$
Hiking-10	373.66 ± 63.78	414.39 ± 73.58	378.64 ± 62.93	385.74 ± 61.30	325.13 ± 65.63	181.65 ± 51.50	$\mathbf{67.86 \pm 31.07}$
	train envs: Stacking-3				train envs: Stacking-1234		
Stacking-3	55.97 ± 26.02	80.80 ± 38.70	29.29 ± 15.58	114.96 ± 37.90	43.35 ± 19.50	23.40 ± 12.52	$\mathbf{22.29 \pm 10.96}$
Stacking-5	147.54 ± 49.16	168.47 ± 73.32	136.06 ± 45.22	226.06 ± 64.15	113.71 ± 33.91	$\mathbf{41.02 \pm 17.55}$	41.36 ± 18.70
Stacking-8	267.47 ± 66.48	322.58 ± 88.01	317.98 ± 71.15	341.84 ± 71.79	235.10 ± 63.27	72.77 ± 31.93	$\mathbf{61.44 \pm 28.07}$
	train envs: PandaReach-3				train envs: PandaReach-1234		
PandaReach-3	192.15 ± 66.30	242.49 ± 84.71	119.99 ± 38.16	103.89 ± 53.00	47.77 ± 12.48	70.57 ± 30.16	$\mathbf{40.35 \pm 16.13}$
PandaReach-5	283.06 ± 74.24	222.15 ± 90.92	220.81 ± 61.86	195.31 ± 42.78	96.40 ± 27.72	174.34 ± 50.19	$\mathbf{53.10 \pm 25.06}$
PandaReach-8	384.47 ± 65.41	313.26 ± 95.68	359.77 ± 62.23	440.76 ± 50.25	373 ± 52.69	347 ± 62.67	$\mathbf{125.16 \pm 44.26}$
	train envs: PandaPush-3				train envs: PandaPush-1234		
PandaPush-3	108.06 ± 51.18	105.06 ± 75.34	164.52 ± 39.28	122.25 ± 38.97	114.16 ± 35.12	$\mathbf{22.64 \pm 12.66}$	34.23 ± 16.38
PandaPush-5	221.66 ± 73.59	197.25 ± 35.66	259.24 ± 53.80	243.74 ± 64.36	305.35 ± 69.62	$\mathbf{32.53 \pm 16.25}$	39.18 ± 15.35
PandaPush-8	377.86 ± 60.12	353.47 ± 75.43	378.17 ± 55.00	376.34 ± 65.02	410.25 ± 61.90	67.61 ± 30.93	$\mathbf{62.62 \pm 27.02}$
	train envs: PandaSlide-3				train envs: PandaSlide-1234		
PandaSlide-3	61.78 ± 30.15	92.05 ± 46.24	372.86 ± 73.99	68.66 ± 28.10	38.57 ± 11.99	39.46 ± 18.98	$\mathbf{34.69 \pm 17.38}$
PandaSlide-5	113.08 ± 53.65	156.47 ± 69.96	440.74 ± 52.91	145.43 ± 42.65	106.71 ± 26.51	74.42 ± 29.25	$\mathbf{50.64 \pm 25.41}$
PandaSlide-8	257.03 ± 77.26	249.52 ± 89.30	449.24 ± 45.62	310.81 ± 58.87	295.95 ± 60.06	266.47 ± 63.84	$\mathbf{196.67 \pm 47.36}$
	train envs: PandaStack-3				train envs: PandaStack-1234		
PandaStack-3	146.32 ± 40.27	130.46 ± 55.15	117.07 ± 40.49	103.30 ± 51.75	30.75 ± 15.29	34.12 ± 17.39	$\mathbf{27.53 \pm 13.76}$
PandaStack-5	236.55 ± 55.26	287.26 ± 76.10	229.11 ± 58.39	221.52 ± 69.39	62.60 ± 26.42	137.47 ± 51.21	$\mathbf{45.46 \pm 20.12}$
PandaStack-8	352.95 ± 64.16	295.67 ± 86.04	377.76 ± 71.03	339.82 ± 64.12	184.81 ± 47.94	279.55 ± 74.26	$\mathbf{132.67 \pm 42.02}$

Table 1: Performance comparison for evaluating policies in the benchmarks is averaged over 10 random seeds. The performance is measured by mean number of steps an agent takes to achieve goals, along with standard deviations. PPO, SAC, HIRO, PRL are trained in specific environments with fixed variations. For example, Hanoi-3 denotes a 3-disc 'Tower of Hanoi' task with three discs. ReptilePPO, ReptilePRL and our approach are trained in environments with a range of variations, e.g., Hanoi-1234. The test domain contains the tasks during training as well as the unseen tasks.

number of steps remains within acceptable limits. In a few benchmarks, RetilePRL achieved the best results and ReptilePPO showed the best performance in Hiking-3. These results illustrate that employing a meta-learning training approach and a designed programmatic policy enables the learned policy to generalize effectively when facing unseen tasks. In contrast, PPO, SAC, HIRO and PRL exhibit poor performance in most benchmarks, with their performance varying significantly as the complexity of the benchmarks increases.

The performance of these algorithms in benchmarks characterized by greater complexity is also illustrated in Table 1. The threshold for the maximum number of agent-environment interactions is set to 500. For instance, when PRL is applied to Hanoi-7, the number of action steps required reaches 500, which means the failure of the task. Policies trained by PPO and HIRO demand approximately 400 steps to complete the task. When confronted with previously unseen tasks during training, policies trained with ReptilePPO or ReptilePRL demonstrate a degree of generalization by completing the task in roughly 100 steps. Evidently, these methods outperform the PPO, HIRO and PRL. Nonetheless, it exhibits subpar performance in intricate scenarios such as hiking-10 or PandaReach-8.

Policies trained using our method consistently exhibit commendable performance. With the exception of PandaSlide-8 and PandaStacking-8, the performance of our method remains stable despite changes in the value of the variations in the benchmarks. Given the exponential complexity of Hanoi, the optimal number of actions for hanoi-7 is 127. In our method, the solution for this task requires 151 action steps, showcasing commendable performance compared to other methods. These results indicate that our approach has the ability to capture the state patterns when trained across the tasks in one domain. The designed policy architecture aids in summarizing useful subroutines. By capturing the state patterns and shared subroutines, our method showcases generalization when facing unseen tasks.

Ablation Study We investigate the impact of two ideas: the combination of a meta-learning framework with a designed programmatic policy and the utilization of RNN blocks in policy architecture. The convergence curves of PPO-N, Reptile, ReptilePRL, and our method (Ours) are compared, as illustrated in Figure 5 Each policy is trained in the task domain and is evaluated in a fixed environment respectively. The vertical axis represents the number of steps required for the agent to achieve its objective, while the horizontal axis denotes the number of iterations involving agent-environment interaction episodes.

First of all, the policies learned by PPO-N struggle to achieving convergence in all benchmarks. Combining the results from Table 1, policies trained across tasks tend to exhibit better performance. It suggests that programmatic policies tend to be more effective in learning sharing subroutines for tasks.

According to Figure 5, ReptilePRL and ReptilePPO achieve similar results in most benchmarks. However, compared to ReptilePPO, ReptilePRL performs more consistent

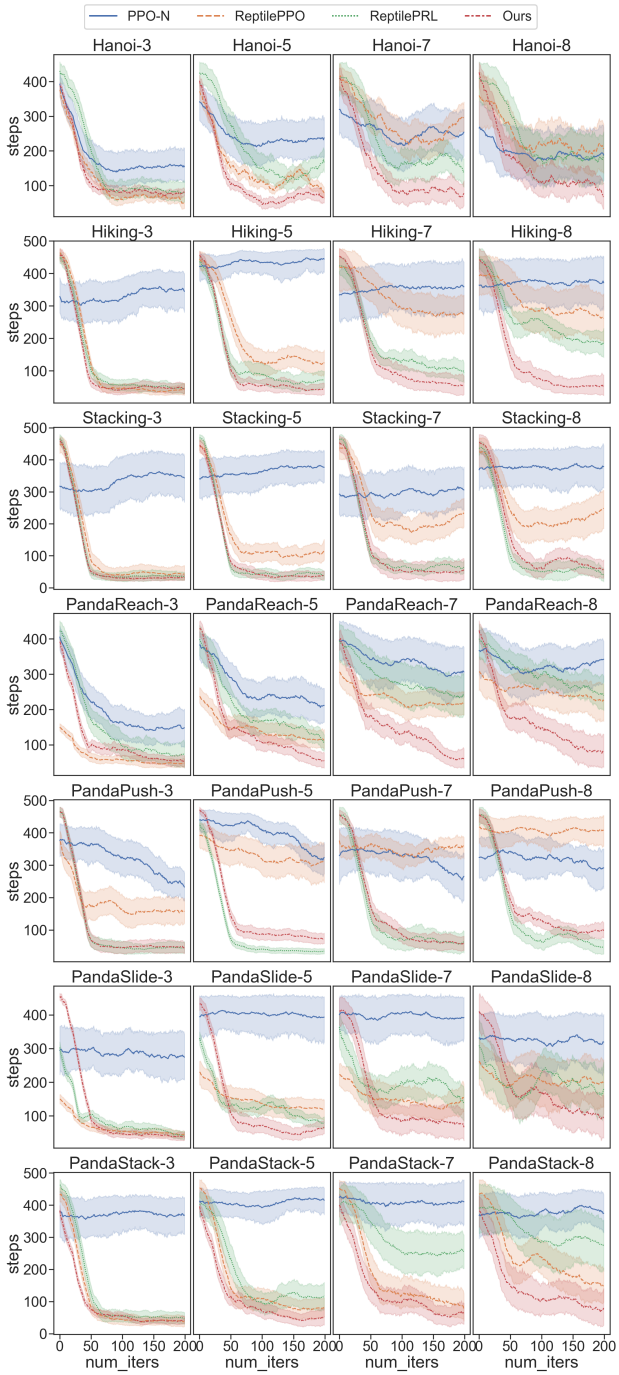


Figure 5: Comparison of the training curve for **PPO-N**, **ReptilePPO**, **ReptilePRL** and **Ours** in the seven benchmarks. Results are averaged over 10 random seeds.

performance across all benchmarks. For example, ReptilePPO struggles to learn to finish the tasks in PandaPush-5, PandaPush-7 and PandaPush-8. This suggests that programmatic policies tend to be more effective to learn sharing subroutines of a task domain. This confirms that a well-designed programmatic policies may be more effective in handling generalization.

In the comparison between our approach and ReptilePRL, both approaches attain commendable performance in most tasks. However, as scenario complexity increases, ReptilePRL struggles to maintain an optimal solution in complex scenarios, whereas our approach consistently converges to a satisfactory solution. This is likely due to the fact that when the agent interacts with environments and preserves the RNN's hidden states between each episode, the agent trained by our method effectively leverages prior knowledge and transfers it to new tasks. These results signify that the incorporation of RNN blocks into programmatic policies effectively improve the ability to generalize across tasks.

4 Related Work

Generalization in RL Generalizing out of data distribution is challenging for reinforcement learning algorithms. A set of research focus on vision-based models to enhance sample efficiency or generalization. One main idea is data augmentation for visual observations such as, random translate, crop, color jitter, patch cutout [Laskin *et al.*, 2020; Yarats *et al.*, 2021]. Moreover, [Hansen and Wang, 2021] proposes SOft Data Augmentation which decouples data augmentation from policy learning.

In robotics, models that perform well in simulators tend to exhibit reduced performance in the real world. Environment randomization is explored to bridge this gap [Tobin *et al.*, 2017]. [Packer *et al.*, 2018] claims that environment randomization is the most effective method so far to improve generalization ability based on experiments on several sets of MuJoCo. [Akkaya *et al.*, 2019] introduces the Automatic Domain Randomization (ADR) algorithm to address the issue of models trained in simulated environments performing poorly in real environments. [Tzeng *et al.*, 2020] treats the transition from simulator to reality as a transfer learning problem. The real world robotic controller is learned by the ideas of domain adaptation and paired image alignment. These methods use environment randomization for sim2real problem. CoinRun, designed to test the generalization performance of deep reinforcement learning algorithms, is introduced by [Cobbe *et al.*, 2019]. An approach, randomizing convolutional neural netowork, is evaluated in CoinRun [Lee *et al.*, 2019]. A regularization parameter is proposed as a positive role to improve the model's generalization. However, there are potential issues with increasing environment randomization, including: increased complexity of the environment, increased complexity of training and dramatically increased variance.

Another way to improve generalization is regularization. According to [Liu *et al.*, 2019; Farebrother *et al.*, 2018], it is claimed that L2 regularization can produce better results than entropy regularization, and L2 regularization can find a good balance point for model's ability and generalization. [Lu *et al.*, 2020] considers deep reinforcement learning models as two parts: the perception layer and the decision-making layer. An information Bottleneck approach is proposed to constrain the information transmitting, due to the perception layer being more prone to overfitting to the current training environment.

Programmatic Policy There existing prior work that mentions programmatic policies to address motion control problems in reinforcement learning. [Trivedi *et al.*, 2021] proposes that combine learning a program embedding space with unsupervised searching to yield a program that maximizes the return of a given task. Besides, the searching part is improved in [Liu *et al.*, 2023]. Though, the program in this work is in the format of a procedure instead of a policy in reinforcement learning, which can not interact with environments.

Some approaches are proposed to learn programmatic policies from neural network based policies, such as learning decision tree policy by model compression [Bastani *et al.*, 2018], learning a program by Bayesian optimization [Verma *et al.*, 2018] or generating symbolic policy by an autoregressive recurrent neural network [Landajuela *et al.*, 2021]. The main idea of these work is to learn programmatic policies by referring to a neural network.

An algorithm for learning programmatic state machine policies that can capture repeating behaviors is proposed in [Inala *et al.*, 2020]. This method follows a teacher-student learning paradigm. Specifically, a programmatic state machine policy is guided by the supervision of a neural network policy trained by RL. Besides the training algorithm, we focus on the generalizability of the model when the variations of the environment change, such as in number or position. [Liu *et al.*, 2018; Cui and Zhu, 2021] propose to encode program architecture search as learning the probability distribution over all possible program derivations induced by a context-free grammar. This allows the search algorithm to efficiently prune away unlikely program derivations to synthesize optimal program architectures. In [Qiu and Zhu, 2022], a programmatic policy is learned by architecture search for control problems without demonstrations. Follow these lines of work, we utilize meta-learning algorithm to synthesize a programmatic policy for generalization in a task domain.

5 Conclusion and Future Work

We introduce an approach aimed at synthesizing programmatic policies that can capture subroutines across tasks in one task domain. In order to improve the effectiveness of training, we utilize a meta-learning algorithm to synthesize the programmatic policy. Additionally, we incorporate RNN blocks for the policy architecture, enabling the utilization of previously acquired knowledge. Experimental results demonstrate the effectiveness of our method in synthesizing a programmatic policy capable of generalizing across tasks, even extending to unseen tasks during training.

Building upon our current method, two avenues for future research emerge. The first involves expanding the applicability of our method to more intricate scenarios, such as the scope of the task domain getting larger. The second direction entails exploring the inclusion of additional grammars into the DSL, resulting in a programmatic policy endowed with a more intricate architecture capable of addressing more complex tasks.

References

[Akkaya *et al.*, 2019] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[Bastani *et al.*, 2018] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2499–2509, 2018.

[Cobbe *et al.*, 2019] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[Cui and Zhu, 2021] Guofeng Cui and He Zhu. Differentiable synthesis of program architectures. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 11123–11135, 2021.

[Farebrother *et al.*, 2018] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[Gallouédec *et al.*, 2021] Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, and Liming Chen. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

[Haarnoja *et al.*, 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[Hansen and Wang, 2021] Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 13611–13617. IEEE, 2021.

[Inala *et al.*, 2020] Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[Landajuela *et al.*, 2021] Mikel Landajuela, Brenden K. Petersen, Sookyung Kim, Cláudio P. Santiago, Ruben Glatt, T. Nathan Mundhenk, Jacob F. Pettit, and Daniel M. Faissol. Discovering symbolic policies with deep reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 5979–5989. PMLR, 2021.

[Laskin *et al.*, 2020] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[Lee *et al.*, 2019] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv preprint arXiv:1910.05396*, 2019.

[Liu *et al.*, 2018] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.

[Liu *et al.*, 2019] Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization–an empirical study on continuous control. *arXiv preprint arXiv:1910.09191*, 2019.

[Liu *et al.*, 2023] Guan-Ting Liu, En-Pei Hu, Pu-Jen Cheng, Hung-Yi Lee, and Shao-Hua Sun. Hierarchical programmatic reinforcement learning via learning to compose programs. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 21672–21697. PMLR, 2023.

[Lu *et al.*, 2020] Xingyu Lu, Kimin Lee, Pieter Abbeel, and Stas Tiomkin. Dynamics generalization via information bottleneck in deep reinforcement learning. *arXiv preprint arXiv:2008.00614*, 2020.

[Nachum *et al.*, 2018] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3307–3317, 2018.

[Nichol and Schulman, 2018] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2(3):4, 2018.

[Packer *et al.*, 2018] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

[Qiu and Zhu, 2022] Wenjie Qiu and He Zhu. Programmatic reinforcement learning without oracles. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Silver and Chitnis, 2020] Tom Silver and Rohan Chitnis. Pddlgym: Gym environments from pddl problems. In *International Conference on Automated Planning and Scheduling (ICAPS) PRL Workshop*, 2020.

[Tobin *et al.*, 2017] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 23–30. IEEE, 2017.

[Trivedi *et al.*, 2021] Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J. Lim. Learning to synthesize programs as interpretable and generalizable policies. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 25146–25163, 2021.

[Tzeng *et al.*, 2020] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. Adapting deep visuomotor representations with weak pairwise constraints. In *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, pages 688–703. Springer, 2020.

[Verma *et al.*, 2018] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5052–5061. PMLR, 2018.

[Verma *et al.*, 2019] Abhinav Verma, Hoang Minh Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15726–15737, 2019.

[Winskel, 1993] Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT press, 1993.

[Yarats *et al.*, 2021] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.