Programmatic Imitation Learning From Unlabeled and Noisy Demonstrations

Jimmy Xin, Linus Zheng, Kia Rahmani, Jiayi Wei, Jarrett Holtz, Isil Dillig, and Joydeep Biswas

***Abstract*—Imitation Learning (IL) is a promising paradigm for teaching robots to perform novel tasks using demonstrations. Most existing approaches for IL utilize neural networks (NN), however, these methods suffer from several well-known limitations: they 1) require large amounts of training data, 2) are hard to interpret, and 3) are hard to refine and adapt. There is an emerging interest in *Programmatic Imitation Learning* (PIL), which offers significant promise in addressing the above limitations. In PIL, the learned policy is represented in a programming language, making it amenable to interpretation and adaptation to novel settings. However, state-of-the-art PIL algorithms assume access to action labels and struggle to learn from noisy real-world demonstrations. In this paper, we propose PLUNDER, a novel PIL algorithm that addresses these shortcomings by synthesizing *probabilistic* programmatic policies that are particularly well-suited for modeling the uncertainties inherent in real-world demonstrations. Our approach leverages an EM loop to simultaneously infer the missing action labels and the most likely probabilistic policy. We benchmark PLUNDER against several established IL techniques, and demonstrate its superiority across five challenging imitation learning tasks under noise. PLUNDER policies outperform the next-best baseline by 19% and 17% in matching the given demonstrations and successfully completing the tasks, respectively.**

***Index Terms*—Computer science, formal languages, machine learning, programming, representation learning, robot learning, robot programming.**

I. Introduction

IMITATION Learning (IL) is a popular approach for teaching robots how to perform novel tasks using only human demonstrations, without the need to specify a reward function or a system transition function [1]. Most current IL approaches use neural networks to represent the learned policy, mapping the agent's state to actions. While effective, these approaches have several well-known limitations: they require large amounts of training data [2]; they are opaque and hard to interpret [3]; and they are hard to refine and adapt to novel settings [4]. Seeking to address these limitations, *programmatic imitation learning* (PIL) approaches synthesize *programmatic policies* from demonstrations, which are human-readable, require significantly fewer demonstrations, and are amenable to refinement and adaptation [5], [6], [7].

Manuscript received 9 November 2023; accepted 14 March 2024. Date of publication 5 April 2024; date of current version 17 April 2024. This letter was recommended for publication by Associate Editor A. M. Ghalamzan Esfahani and Editor A. Faust upon evaluation of the reviewers' comments. This work was supported by the National Science Foundation, under Grant CAREER-2046955, Grant CCF-2319471, Grant CCF-1762299, Grant CCF-1918889, Grant CNS-1908304, Grant CCF-1901376, Grant CNS-2120696, and Grant CCF-2210831. *(Jimmy Xin and Linus Zheng contributed equally to this work.) (Corresponding author: Kia Rahmani.)*

Jimmy Xin, Linus Zheng, Kia Rahmani, Jiayi Wei, Isil Dillig, and Joydeep Biswas are with the University of Texas at Austin, Austin, TX 78712 USA (e-mail: kiar@utexas.edu).

Jarrett Holtz is with Robert Bosch LLC, Farmington Hills, MI 48331 USA.

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3385691, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3385691

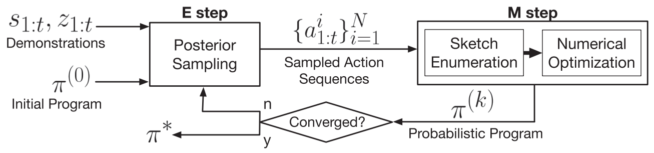


Fig. 1. Overview of PLUNDER.

However, existing PIL methods require *action labels* for human demonstrations, which are often unavailable in real-world settings. Furthermore, these methods assume that the demonstrations are *nearly noise-free*, which is rarely the case in practice. In this paper, we introduce two key insights to address the above challenges. First, given human demonstrations without action labels, inferring the action labels is a *latent variable estimation* problem. Second, instead of synthesizing a deterministic policy, synthesizing a *probabilistic* policy allows us to model the uncertainties inherent in real-world demonstrations [8]. Combining these insights, we introduce PLUNDER, a new PIL algorithm that synthesizes *probabilistic programmatic policies* from unlabeled and noisy demonstrations.

Fig. 1 presents an overview of PLUNDER. The algorithm starts with an initial randomized policy ($\pi^{(0)}$) and iteratively improves this policy while also improving its estimates of the action labels using an Expectation-Maximization (EM) algorithm [9]. In the E step, PLUNDER samples posterior action label sequences by combining the current policy with the given demonstrations. In the M step, it synthesizes a new policy that maximizes the likelihood of producing actions that match the previously sampled action labels. To improve the scalability of the M step, we also propose an incremental synthesis technique that narrows the search on policies similar to the best policy found in the previous iteration. This process is repeated until convergence, yielding jointly an optimal policy π^* and the most likely action labels for the demonstrations.

To evaluate our approach, we apply PLUNDER to five standard imitation learning tasks and compare the results with multiple baselines, including four state-of-the-art IL techniques. We empirically show that PLUNDER synthesizes policies that conform to the demonstrations with 95% accuracy, which is 19% higher than the next best baseline. Furthermore, PLUNDER's policies are 17% more successful than the closest baseline in completing the tasks.

In summary, the key contribution of this paper is PLUNDER, which, to the best of our knowledge, represents the first *probabilistic PIL* approach specifically designed to synthesize programmatic policies from unlabeled and noisy demonstrations. We validate our approach through extensive empirical evaluation on challenging IL benchmarks. Our implementation of the algorithm and the results of our empirical evaluation are available on the project website for PLUNDER (https://amrl.cs.utexas.edu/plunder/).

II. Related Work

Imitation Learning: Imitation Learning (IL) provides a promising framework for learning autonomous agent behaviors from human demonstrations [1]. Unlike Reinforcement Learning (RL), IL does not require explicit reward signals. Although RL has been successful in addressing numerous complex tasks [10], IL techniques are better suited for many domains where formulating effective reward objectives proves extremely challenging [11].

Within the broad umbrella of IL techniques, Behavior Cloning (BC) and Inverse Reinforcement Learning (IRL) have gained notable attention. BC focuses on establishing a function that directly associates observations with actions [12], [13], [14], [15]. In contrast, IRL aims to decode the intrinsic reward structure from the given trajectories and subsequently leverages RL for policy inference [16]. IRL typically demands more sophisticated algorithms and a substantial amount of training data to infer the expert's inherent motivations. Its primary objective is to *enhance* the expert's performance or *transfer* the acquired knowledge to related tasks [17]. In contrast, our approach in PLUNDER aligns more closely with the BC setup, where a rapid and precise replication of expert behaviors is needed, and access to a complete simulation of the system is not available.

Programmatic Imitation Learning: There is a growing interest in machine learning methodologies with enhanced *interpretability* [18], meaning that the learned models can be expressed using programmatic and human-readable structures like decision trees [7] and finite-state machines [19], [20]. For sequential decision-making tasks, programmatic policy inference has been studied under both RL and IL settings [3]. Programmatic Imitation Learning (PIL) aims at learning such interpretable policy representations from a set of expert demonstrations [5], [21], [22].

One of the early PIL approaches was introduced in [23], which involves learning a human-readable plan from a single real-world demonstration. This plan is converted into a sequence of robotic actions suitable for broader applications. However, these plans provide only a basic description of the necessary state sequence for task completion. In contrast, the programs generated by PLUNDER are more informative, explaining the specific conditions in the state space that trigger an action, and incorporate details on noise at decision boundaries.

A more recent PIL method is LDIPS [5], that attempts to synthesize programmatic policies from human demonstrations. Similar to PLUNDER, LDIPS uses a domain-specific language to enumerate program sketches, but it relies on a Satisfiability Modulo Theories (SMT) solver to find sketch completions. Moreover, LDIPS requires access to action labels and only synthesizes deterministic policies incapable of reasoning about noise in the demonstrations. This leads to lower performance, as shown by our experiments.

Finally, PROLEX [6] is a recent PIL method for long-horizon tasks in complex environments. PROLEX synthesizes complex policies with nested loops and uses a Large Language Model (LLM) to leverage common-sense relationships between objects and their attributes for scalable synthesis. However, unlike PLUNDER, which learns from low-level and noisy demonstrations, PROLEX is constrained to high-level and symbolic task demonstrations.

Imitation from Observations: PLUNDER is related to a specific subset of IL research that focuses on Imitation from Observations (IfO) [24], [25], [26]. IfO aims to enable learning from existing demonstration resources, such as online videos of humans performing a wide range of tasks [27]. These resources provide information about the state of the environment, but they do not include the specific actions executed by the demonstrators. For instance, a state-of-the-art IfO approach is GAIfO [24], which aims to infer the state-transition cost function of an expert by concurrently training a policy network and a discriminator network through a generative adversarial mechanism [28]. However, the neural policies trained by GAIfO are opaque and lack interpretability, and as such, do not effectively address the PIL problem solved by PLUNDER. Furthermore, GAIfO and similar neural network-based approaches often require significantly more training data than PLUNDER to achieve accurate policies.

Expectation Maximization Techniques: The Expectation Maximization (EM) algorithm is a commonly used technique in situations where both the generative model and its hidden states are unknown [9]. Recently, EM has been applied in various robotics domains [29]. For example, in [30], the authors propose using an EM loop to simultaneously learn a dynamics model and estimate the robot's state trajectories. In [31], an internal EM loop was employed to optimize a learned policy. However, this method is focused on RL and is not well-suited for handling noise.

III. Problem Formulation

Given a set of demonstrations of a task, the PIL problem addressed in this paper is to infer a probabilistic *Action Selection Policy (ASP)* that is maximally consistent with the given demonstrations and that captures the behavior intended by the demonstrator.

In particular, we consider learning policies over a continuous state space S (e.g., $\mathbb{SE}(2)$ for a ground mobile robot) and a discrete action space A, where the actions are abstracted as *skills* [32], [33]. For instance, an autonomous vehicle may have skills such as accelerate (ACC), decelerate (DEC), and maintain constant velocity (CON). A probabilistic ASP, π, is a probabilistic program that, given the agent's current state ($s_i \in S$) and its current action ($a_i \in A$), defines the probability of taking the next action $a_{i+1} \sim \pi(s_i, a_i)$.

The given demonstrations consist of an agent's *observations* $z_{1:t}$ (e.g., the vehicle's acceleration input), and the corresponding *state* trajectories, $s_{1:t}$ (e.g., the vehicle's velocity), but do not include any action labels. Our objective is to infer a *maximum a posteriori (MAP)* estimate of an ASP, as $\pi^* = \arg\max_\pi P(z_{1:t}, \pi | s_{1:t})$, which can be factorized as $\pi^* = \arg\max_\pi P(z_{1:t}|s_{1:t}, \pi)P(\pi)$. We next introduce action labels, $a_{1:t}$, as marginalized latent variables which should be estimated jointly with the ASP:

$$\pi^* = \arg\max_\pi \sum_{a_{1:t}} P(z_{1:t}|a_{1:t}, s_{1:t}) P(a_{1:t}|s_{1:t}, \pi) P(\pi) \tag{1}$$

In the above formula $P(z_{1:t}|a_{1:t}, s_{1:t})$ is the *observation model* that defines the likelihood of an observation trajectory given the agent's actions and states, and $P(a_{1:t}|s_{1:t}, \pi)$ is defined by the policy π. Function $P(\cdot)$ denotes a prior distribution on ASPs. A specific instance of $P(\cdot)$ will be introduced in Section IV, which aims to reduce overfitting to the data. We also make the Markov assumptions on ASPs and the observation model, i.e., $P(a_{1:t}|s_{1:t}, \pi) = \prod_{i=1}^{t} P(a_i|a_{i-1}, s_i, \pi)$, and $P(z_{1:t}|a_{1:t}, s_{1:t}) = \prod_{i=1}^{t} P(z_i|a_i, s_i)$, yielding our final objective:

$$\pi^* = \arg\max_\pi \sum_{a_{1:t}} \prod_{i=1}^{t} P(z_i|a_i, s_i)\, P(a_i|a_{i-1}, s_i, \pi)\, P(\pi). \tag{2}$$

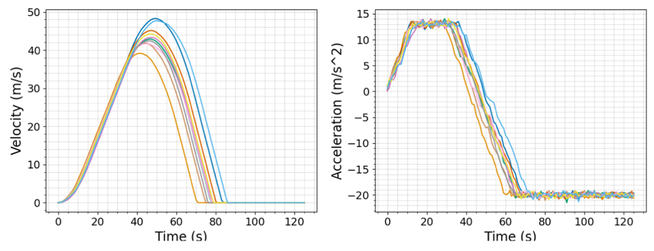
There are two major challenges in solving (2). First, directly computing the sum over all action trajectories is computationally intractable. Second, the search space for probabilistic policies is prohibitively large, making any naïve search within this space unscalable. In Section IV, we explain our approach to overcome both of these challenges in a tractable manner.

A. Example: Stop-Sign

We now illustrate our approach using the example of an autonomous vehicle tasked with driving along a straight road and making appropriate stops at each stop sign. Defining a reward function or an objective function that accurately captures the desired behaviors for executing this task is challenging. Instead, we aim to learn an ASP from expert demonstrations of the task using imitation learning.

Fig. 2 presents 10 demonstrations for this task. Each demonstration consists of the vehicle's velocity and acceleration trajectories between two consecutive stop signs. There are noticeable variations in these demonstrations. For instance, some demonstrations prefer a more gradual acceleration and deceleration for a comfortable ride. In contrast, others showcase harder acceleration and deceleration.

As we will discuss in the subsequent sections, the PLUNDER algorithm is capable of reasoning about variations and noise in demonstrations and can synthesize a probabilistic ASP that captures the behavior intended by the demonstrations collectively.



(a) velocity/time (b) acceleration/time

Fig. 2. Demonstration trajectories for the Stop Sign task. The acceleration value of this particular vehicle cannot exceed $a_{\max} \approx 13\,\mathrm{m/s^2}$ or drop below $a_{\min} \approx -20\,\mathrm{m/s^2}$.

Algorithm 1: PLUNDER Algorithm.

Inputs: state trajectories: $\bar{S}$, observation trajectories: $\bar{Z}$, observation model: O, initial policy: $\pi^{(0)}$, convergence threshold: γ

Output: probabilistic action selection policy: π^*

```
1: $k = 0$
2: while likelihood($\bar{S}, \bar{Z}, O, \pi^{(k)}$) $\leq \gamma$ do
3:     $\{a^i_{1:t}\}_{i=1}^N$ := runPF($\bar{S}, \bar{Z}, O, \pi^{(k)}$) // E step
4:     $\pi^{(k+1)}$ := synthesize($\pi^{(k)}, \bar{S}, \{a^i_{1:t}\}_{i=1}^N$) // M step
5:     $k = k + 1$
6: end while
8: return $\pi^{(k)}$
```

IV. THE PLUNDER ALGORITHM

Algorithm 1 provides an overview of our Expectation-Maximization (EM) approach, which is designed to tractably approximate the MAP estimate in (2). The inputs to the algorithm are a set of D demonstrations consisting of state trajectories ($\bar{S} := \{s^i_{1:t}\}_{i=1}^D$) and observation trajectories ($\bar{Z} := \{z^i_{1:t}\}_{i=1}^D$), the observation model O, the initial policy $\pi^{(0)}$, and a threshold, γ, for determining whether the synthesized policy has converged.

The algorithm alternates between the E step (line 3) and the M step (line 4). In the E step, a posterior sampling of N action label trajectories is performed using the current candidate ASP $\pi^{(k)}$ and the given demonstration trajectories. These sampled action trajectories are then used in the M step to synthesize a policy that is maximally consistent with those action trajectories. After each iteration of the E and M steps, the likelihood of the synthesized policy's prediction on the given trajectories is measured. Once the desired likelihood is reached (i.e., the policy has converged), the algorithm outputs the latest synthesized policy as its final result, π^*.

In the remainder of this section, we introduce the syntax of probabilistic ASPs synthesized in PLUNDER, and present further details about the E and M steps of Algorithm 1.

A. Probabilistic ASPs

Figure 3 presents a probabilistic Domain-Specific Language (pDSL) [34] used to compose ASPs in PLUNDER. A policy π in

$$\begin{array}{rlcl}
\text{(Feature)} & f & := & y_t \mid c \mid g(f_1, \ldots, f_n) \\
\text{(Prob. Dist.)} & \psi & := & r \mid \texttt{lgs}(f, x_0, k) \\
\text{(Guard)} & \phi & := & \texttt{flp}(\psi) \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
\text{(Trans. Seq.)} & \tau & := & \texttt{if } (\phi \texttt{ and } a_{t-1} = A) \texttt{ then } A' \,; \tau \mid \texttt{skip} \\
\text{(ASP)} & \pi(y_t, a_{t-1}) & := & \tau
\end{array}$$

Fig. 3. Grammar of ASPs. Here, y_t, a_{t-1} are inputs representing the current state and previous action, c is a constant, A is an action, and g is a built-in ($+$, $\times$ etc) or domain-specific feature extraction function (e.g., `timeToStop`).

this DSL takes as input the current state y_t and previous action a_{t-1} and returns a new action a'. A policy is a sequence of conditionals $\texttt{if}(\phi \texttt{ and } a_{t-1} = A) \texttt{ then } A'$, guarding transitions from previous action A to new action A' based on the predicate ϕ that is evaluated on the current state y. For brevity, we use the notation $\phi_{A,A'}$ to denote that ϕ is the guard associated with transition from action A to action A' with the understanding that it would comprise the statement $\texttt{if}(\phi_{A,A'} \texttt{ and } a_{t-1} = A) \texttt{ then } A'$. Thus an ASP can be alternatively viewed as an ordered list of guards $\phi_{A_0,A'_0}, \ldots, \phi_{A_n,A'_n}$.

Guards in this DSL are boolean combinations of probabilistic predicates, parameterized over policy parameters r, x_0, k to be synthesized. Atomic predicates are of the form $\texttt{flp}(\psi)$, which evaluates to true with probability ψ and false with probability $1 - \psi$. In the simplest case, ψ is a constant $r \in [0, 1]$. In most cases, however, ψ is a logistic function $\texttt{lgs}(f, x_0, k)$, such that $\texttt{flp}(\texttt{lgs}(f, x_0, k))$ smoothly approximates the inequality $f \leq x_0$ using the logistic function parameter k that controls the sharpness of the transition. Note that f is either the current state y_t or a feature extracted from the current state using a built-in function, such as `timeToStp` in the Stop-Sign domain. As in other programmatic policy synthesis algorithms [5], [6], [35], the feature extraction functions as well as the space of possible actions are domain-dependent.

B. Expectation (E) Step

During the E step (line 3) of Algorithm 1, the current estimate of the ASP, $\pi^{(k)}$, is used to sample N plausible action sequences from the posterior distribution $\{a^i_{1:t}\}^N_{i=1} \sim P(. \mid s_{1:t}, z_{1:t}, \pi^{(k)})$. This posterior sampling problem is particularly amenable to inference using Monte-Carlo estimation algorithms, and we use a *particle filter* [36] to sample a policy's action label sequences.

The function $\texttt{runPF}(\bar{S}, \bar{Z}, O, \pi^{(k)})$ implements the particle filter as follows. For each state trajectory $s_{1:t} \in \bar{S}$, the function utilizes $\pi^{(k)}$ to sample action labels forward in time. Next, it employs the observation model O to re-weight and re-sample the particles. This ensures that sequences more aligned with the provided observations have a higher likelihood of duplication and representation in the final sample set. After completing this procedure for all provided demonstrations, the final particle set represents $\{a^i_{1:t}\}^N_{i=1}$, i.e., N plausible action label sequences given the current policy and the demonstration traces.

C. Maximization (M) Step

During the M Step (line 4) of Algorithm 1, a new ASP is synthesized using the function $\texttt{synthesize}(.)$. This function, given the state trajectories $\bar{S}$ and the sampled action sequences from the previous E step $\{a^i_{1:t}\}^N_{i=1}$, aims to solve the following objective:

$$\pi^{(k+1)} = \underset{\pi \in \mathbb{N}(\pi^{(k)})}{\arg\max} \sum_{i=1}^{N} \log P(a^i_{1:t} \mid s_{1:t}, \pi) + \log P(\pi) \tag{3}$$

The above formula indicates that the synthesized policy should be as consistent as possible with the sampled action sequences and should have a high probability of occurrence based on the ASP prior. We define the ASP prior as $\log P(\pi) = -\lambda \cdot \text{size}(\pi)$, where $\lambda \geq 0$ is a regularization constant to promote smaller sized ASPs [37]. Larger values of λ discourage the synthesis of overly complex programs that overfit to the demonstrations.

The function $\texttt{synthesize}(.)$ implements a bottom-up inductive synthesizer [37], where the production rules described in Fig. 3 are used to enumerate a set of policy structures, or *sketches*. The enumeration begins with the set of numerical features – consisting of constants, variables, and function applications – which are used to construct probabilistic threshold sketches with two unknown parameters. The threshold sketches are combined using disjunctions and conjunctions to construct the final set of policy-level sketches.

Each policy sketch can be converted into a complete policy by determining the unknown real-valued parameters in the probabilistic thresholds. Our goal is to identify the parameters that optimize the objective given by (3). To solve this optimization problem, we employ the Line Search Gradient Descent (L-BFGS) algorithm [38], selecting the top result out of four random initial assignments. After completing all sketches, the $\texttt{synthesize}(.)$ function returns the optimal policy according to (3) as its final output.

While the above inductive synthesis approach is guaranteed to find the optimal solution within its bounded search space, enumerating programs this way rapidly becomes intractable. This is especially problematic in our setup, where the synthesis is integrated within the EM loop, which might need many iterations to converge. To address this challenge, we restrict the search space for $\pi^{(k+1)}$ to $\mathbb{N}(\pi^{(k)})$, the *syntactic neighborhood* of $\pi^{(k)}$. To enumerate the syntactic neighborhood of a policy, we mutate its abstract syntax tree to derive a set of new expressions. In particular, we apply the following types of mutations:

1) Add a new threshold predicate with a random feature.
2) Simplify the policy by removing an existing predicate.
3) Swap conjunctions with disjunctions, and vice-versa.
4) Augment numerical features by applying random functions with random parameters.
5) Simplify features by removing function applications.

Each syntactic neighborhood, in addition to the above set of mutated expressions, contains a set of atomic features. This allows the synthesizer to *"reset"* to simpler policies if needed.

To further improve the scalability of the $\texttt{synthesize}(.)$ function, we also prune the search space using a type system based on physical dimensional constraints, as previously done in [5]. This type system tracks the physical dimensions of expressions and discards those that are inconsistent.

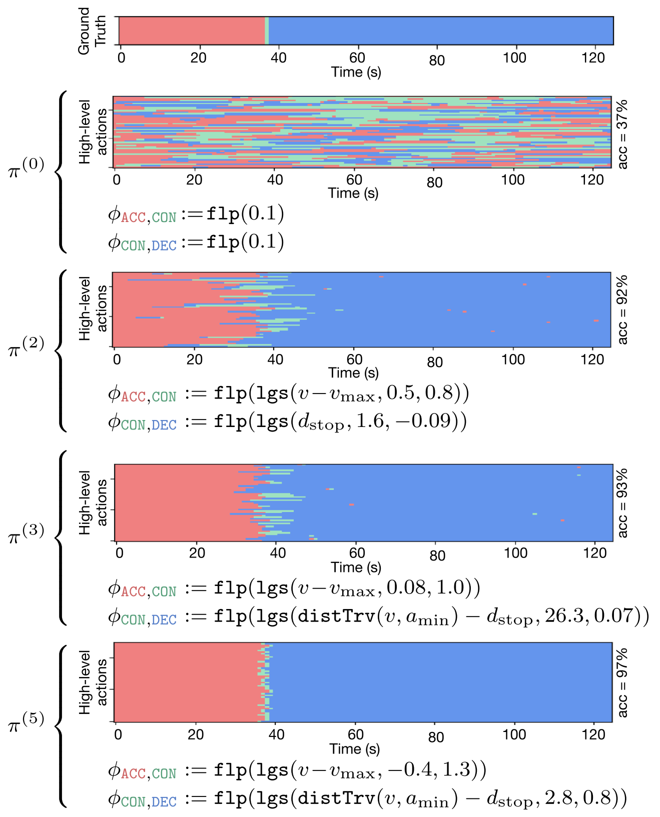


Fig. 4. Best candidate programs found at each iteration and the corresponding action sequence samples. The ground-truth sequence is shown at the top. Only $\phi_{\text{ACC},\text{CON}}$ and $\phi_{\text{CON},\text{DEC}}$ from each policy are shown due to space constraints.

The above two pruning methodologies significantly reduce the policy search space, allowing the $\texttt{synthesize}(.)$ function to be tractably used within our EM framework.

D. Example: Stop-Sign

Figure 4 displays the results of four non-consecutive iterations of the EM loop for the Stop-Sign example. Each iteration presents $\pi^{(i)}$, the candidate policy synthesized during the M step, and its accuracy, alongside the corresponding action sequences sampled using the policy. Due to space constraints, we only show the conditions $\phi_{\text{ACC},\text{CON}}$ and $\phi_{\text{CON},\text{DEC}}$ for each candidate policy. These conditions represent transitions from acceleration to constant velocity, and from constant velocity to deceleration, respectively. Each sampled action sequence is depicted as a color-coded horizontal line. Specifically, ACC time-steps are shown in red, CON time-steps are shown in cyan, and DEC time-steps are shown in blue. Because the candidate policies are probabilistic, the sequences from the same policy exhibit random variations. However, these variations diminish as the EM loop progresses and confidence in the synthesized policy increases.

The EM loop is initiated with a trivial ASP, $\pi^{(0)}$, employing $\texttt{flp}(0.1)$ for all transition conditions. The algorithm then alternates between E and M Steps, progressively refining the candidate policy. This refinement continues until the desired level of accuracy is achieved in $\pi^{(5)}$, which is returned as the final solution for the Stop-Sign task.

In $\pi^{(5)}$, the transition condition $\phi_{\text{ACC},\text{CON}}$ specifies that the vehicle should switch from acceleration to constant velocity, as its velocity nears the recommended maximum velocity, $v_{\max}$. Likewise, $\phi_{\text{CON},\text{DEC}}$ indicates that the vehicle should transition from constant speed to deceleration when the estimated stop distance based on the current speed ($\texttt{distTrv}$), approaches the remaining distance to the next stop sign (d_{stop}). These probabilistic expressions intuitively encapsulate the human demonstrators' intent for this specific task and are amenable to fine-tuning [39].

Synthesizing ASPs such as $\pi^{(5)}$, is challenging due to their intricate structures with multiple functions, variables, and numerical constants. In the subsequent section, we will present more complex examples featuring multiple disjuncts and conjuncts drawn from our benchmarks.

V. Experimental Evaluations

We evaluated our approach using five standard imitation learning tasks from two challenging environments.

Autonomous Vehicle Environment: We utilize the open-source simulation environment HIGHWAY-ENV [40], where the demonstrator controls the acceleration and steering of a vehicle moving on a straight multi-lane highway. Policies have access to the position and velocity of the controlled vehicle and vehicles in adjacent lanes, and may select from action labels such as `accelerate`, `decelerate`, `turn_left`, and `turn_right`. We consider the following three tasks within this environment. The **Stop Sign (SS)** task is the motivating example described in previous sections. The **Pass Traffic (PT)** task requires appropriately selecting and switching lanes in the highway traffic. In the **Merge (MG)** task, the vehicle begins in the leftmost lane and must merge into the rightmost lane without colliding with traffic.

Robotic Arm Environment: We also use the open-source PANDA-GYM framework [41] that simulates a multi-joint robotic arm. Within this environment, the demonstrator manages both the end effector's movement in three-dimensional space and the signal to control its grip. Policies have access to the arm, object, and target locations, in three dimensions, and may select from action labels such as moving to an object, moving to the target, grasping an object, and lifting the arm. We evaluate our approach using two tasks defined in this environment. The first is the **Pick and Place (PP)** task, which requires the arm to grab an object and then move it to a designated location in an uncluttered environment. The second is the **Stack (ST)** task, which is more complex and requires stacking two boxes on top of each other in the correct order.

Ground-truth Demonstrations: For each of the above five tasks, we generated a set of 20 to 30 demonstrations using a ground-truth policy that we implemented manually. The demonstrations for each task were equally divided into a training set and a test set. Fig. 2 presented the training set for the SS task. In addition to the *transition noise* caused by the probabilistic nature of the ground-truth policies, the generated trajectories were also perturbed with additive Gaussian *actuation noise*.

Human-generated Demonstrations: In order to evaluate the baselines' ability to learn tasks under more realistic conditions, we also created datasets of human-generated demonstrations for PT and MG tasks using a dual-joystick controller for vehicle steering and acceleration. We refer to these datasets as **(PT-H)** and **(MG-H)**. These demonstrations have actuation noise similar to that of the ground truth demonstrations, and they exhibit higher transition noise due to the inherent inaccuracies of human behavior.

A. Baselines

We compare the performance of PLUNDER against six baselines, including four state-of-the-art IL approaches.

Greedy: Our first baseline represents a straightforward solution to the problem of imitation learning from unlabeled trajectories, where each time step is greedily labeled with the action whose likelihood is highest according to the observation model. The same policy synthesizer as PLUNDER's M step is then applied on these greedily labeled trajectories.

OneShot: The next baseline synthesizes a policy from labels sampled using a particle filter and the $\pi^{(0)}$ policy. This method can be viewed as the first iteration of PLUNDER, except it does not employ incremental synthesis, and instead directly searches over a larger program space in one shot.

LDIPS: Our third baseline is LDIPS [5], a state-of-the-art PIL approach which synthesizes a policy via sketch enumeration and a reduction to satisfiability modulo theories (SMT). Unlike PLUNDER, LDIPS does not account for noise in the demonstrations and generates only deterministic policies. Furthermore, LDIPS requires action labels for synthesizing a policy. For this, we use the same action labels given to the OneShot baseline.

BC/BC+: The next baseline (BC) implements Behavior Cloning [42], a well-established imitation learning technique. We trained a fully-connected feed-forward neural network with 3 layers to predict the next observation conditioned on the previous state. BC+ extends the BC baseline by incorporating access to action labels and the observation model. In BC+, rather than predicting observations directly, the network outputs a distribution over the available action labels. The observations are then predicted by performing a weighted sum based on the output of the observation model for each action label.

GAIfO: Our next baseline is GAIfO [24], a state-of-the-art method specifically designed for imitation learning from unlabeled demonstrations, using the principles of Generative Adversarial Networks (GANs) [28].

Behavior Transformers: Our final baseline is Behavior Transformers (BeT) [43], a recent method that employs transformer-based sequence models for imitation learning from a given set of continuous observation and action pairs. BeT is capable of multi-modal learning and provides support for both Markovian and non-Markovian settings.

B. Alignment With Demonstrations

We evaluate the ability of PLUNDER and the baselines to learn a policy that closely mimics the ground-truth demonstrations in the training set. Fig. 5 presents the accuracy of action labels each learned policy produces on the test trajectories. BC and BeT are excluded because they do not produce action labels. We also include results from the ground-truth policy (GT).

The results reveal that PLUNDER consistently outperforms all other baselines, achieving an average accuracy of 95%, which is only 2.7% below that of the ground truth policy. The Greedy approach performs particularly poorly because of its tendency to over-fit to noise. The results confirm that utilizing a particle filter for inferring action labels, as implemented in the OneShot baseline, yields better performance compared to the naïve Greedy approach. We also find that the performance of LDIPS is significantly lower than that of PLUNDER, confirming that deterministic policies are insufficient for capturing uncertainties present in the given demonstrations.

We also report the log-likelihood of observations generated by each policy in Fig. 6. Again, across all evaluated tasks, PLUNDER exhibits superior performance, yielding the highest log-likelihood.

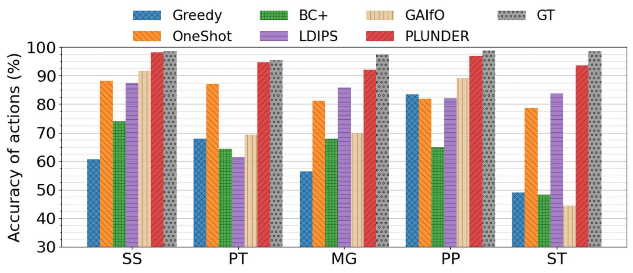


Fig. 5. Accuracy of action labels.

	SS	PT	MG	PP	ST	avg
Greedy	-2.19	-2.10	-2.41	-0.51	-5.03	-2.45
OneShot	-1.12	-0.47	-1.32	-1.54	-1.64	-1.22
BC	-3.47	-0.73	-1.16	-1.22	-1.50	-1.62
BC+	-1.20	-0.69	-1.12	-0.12	-1.53	-0.93
GAIfO	-1.19	-1.30	-1.63	-0.28	-8.82	-2.64
LDIPS	-1.40	-2.03	-1.25	-1.75	-1.29	-1.54
BeT	-2.18	-2.34	-1.14	-1.68	-2.93	-2.054
PLUNDER	**-0.78**	**0.37**	**-0.89**	**0.59**	**-0.96**	**-0.33**
GT	-0.79	0.53	-0.69	0.66	-0.69	-0.196

Fig. 6. Log-likelihood of observations.

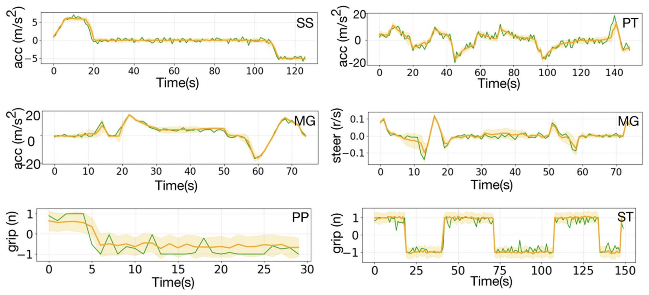
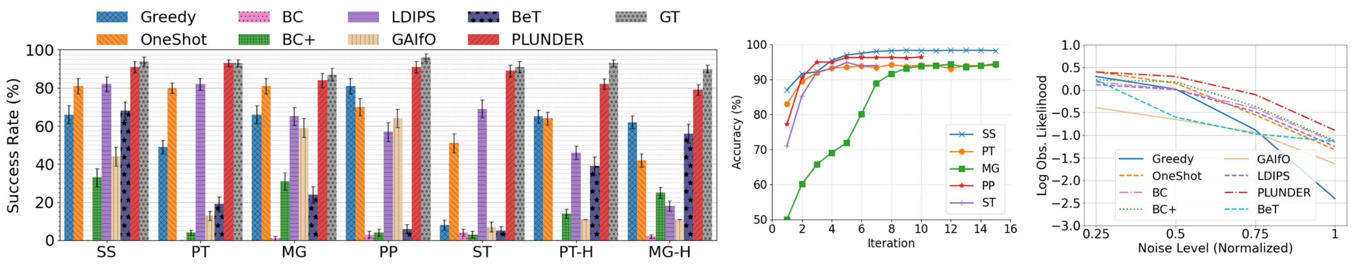


Fig. 7. Observation trajectories from the ground-truth policy (green line) and the PLUNDER policy (orange line).



(a) Success Rate (b) Convergence (c) Noise Tolerance

Fig. 8. Empirical results: (a) the success rates of policies learned across all baseline approaches, (b) the progression of the EM loop until convergence, and (c) the impact of noise on the learning performance of each baseline approach.

Fig. 7 visualizes samples of the ground-truth trajectories (in green) and the corresponding trajectories generated by the PLUNDER policy (in orange). The trajectories synthesized by PLUNDER closely align with the ground-truth across all benchmarks. Due to space limitations, we include a subset of observations from each task; however, we witnessed similar behaviors across all observations.

C. Task Completion Rate

We next evaluate the success rate of the learned policies using PLUNDER and other baselines. Each learned policy was executed 100 times on randomly initialized environments and we report the percentage of executions where the task was successfully completed.

The success rates of each baseline, with 1 SD error bars, are presented in Fig. 8(a). PLUNDER achieves an average success rate of 90%, the highest among all baselines. In contrast, the BC and BC+ approaches perform particularly poorly. We attribute this to the compounding errors effect and the inherent inability of BC methods to generalize from a small training set. Notably, the gap between PLUNDER and all other baselines is larger in PT-H and MG-H with more realistic demonstration datasets.

D. Convergence

Our experiments support our hypothesis that the accuracy of the synthesized policies increases with each iteration of the EM loop. Fig. 8(b) illustrates the performance of the synthesized policy after each EM loop iterations. PLUNDER converges to a policy across all tasks in fewer than 10 iterations.

E. Impact of Noise

We evaluate the performance of each baseline as we vary the amount of actuation noise in the training data for the MG task. The observation log likelihoods of the trajectories produced by all methods are depicted in Fig. 8(c). All approaches show a decrease in performance as the noise increases; however, PLUNDER consistently surpasses other baselines, indicating its superior robustness against noise. We also note that OneShot is more robust against noise compared to Greedy. This underscores the significance of employing a particle filter, even when backed by a simplistic prior. Lastly, the results indicate that BC+ has an advantage over BC, hinting that having access to the observation model provides useful inductive bias to overcome the noise in the data.

TABLE I
AST SIZE OF SYNTHESIZED POLICIES

Benchmark	SS	PT	MG	PP	ST	PT-H	MG-H
AST Size	25	96	46	22	55	196	98

F. Analysis

Table I shows the number of nodes in the AST (Abstract Syntax Tree) of the synthesized policies for each task. We found that in all tasks, the synthesized policies contain non-trivial conditions that require multiple incremental synthesis steps to be discovered.

For instance, the following transition condition was synthesized in the PT task:

$$\begin{aligned}\phi_{\texttt{FASTER,LANE\_LEFT}} = {} & \texttt{flp}(\texttt{lgs}(x - f_x, -30.62, 1.78)) \\ & \wedge\ \texttt{flp}(\texttt{lgs}(f_x - l_x, -4.92, -6.53)) \\ & \wedge\ \texttt{flp}(\texttt{lgs}(r_x - l_x, 2.95, -1.67))\end{aligned}$$

The expression above accurately reflects the behaviors observed in the demonstrations. It states: "If the vehicle's position (x) is approaching the position of the vehicle in front (f_x), and the position of the vehicle to the left (l_x) is farther away than both the vehicle in front and the position of the vehicle to the right (r_x), then switch to the left lane". Such expressions are amenable to refinement and formal verification, which is a significant advantage of PLUNDER over methods such as BC, BC+, GAIfO, and BeT. Note that expressions of this complexity exceed the capabilities of naïve enumeration techniques, which would need to optimize real-valued parameters across more than a billion sketches to identify such a policy. Such enumeration would not be feasible within an EM loop.

VI. CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

We presented PLUNDER, an algorithm for inferring probabilistic programs from noisy and unlabeled demonstrations. A reusable implementation of this algorithm, as well as all empirical evaluation results, are made available online. Our approach encounters the typical limitations associated with syntax-based enumerative program synthesis algorithms, namely, the exponential growth in the search space as the size of the DSL and the complexity of tasks increase. In PLUNDER, we achieved tractable synthesis through careful design of the DSL and the application of a series of heuristics. Additionally, the EM framework is generally susceptible to local optima. In PLUNDER, we reduce the risk of encountering such local solutions by performing multiple runs of the synthesizer from random initial seeds, and by incorporating simple atomic policies outside the syntactic neighborhood of the current policy. For future work, we plan to devise a more general solution by leveraging recent advancements in program synthesis algorithms, such as neural-guided program search [44] and those utilizing Large Language Models [6], and apply these solutions to real-world robot data. We also aim to minimize the required user input by optimizing the observation model concurrently, and introduce an automated parameter adjustment mechanism for the hyper-parameter λ.

ACKNOWLEDGMENT

The authors would like to thank our anonymous reviewers for their helpful and insightful feedback. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] B. Zheng, S. Verma, J. Zhou, I. W. Tsang, and F. Chen, "Imitation learning: Progress, taxonomies and challenges," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Oct. 25, 2022, doi: 10.1109/TNNLS.2022.3213246.
[2] N. Sünderhauf et al., "The limits and potentials of deep learning for robotics," *Int. J. Robot. Res.*, vol. 37, no. 4/5, pp. 405–420, 2018.
[3] N. Topin and M. Veloso, "Generation of policy-level explanations for reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 2514–2521.
[4] Y. Chebotar et al., "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 8973–8979.
[5] J. Holtz, A. Guha, and J. Biswas, "Robot action selection learning via layered dimension informed program synthesis," in *Proc. Conf. Robot Learn.*, 2020, pp. 1471–1480.
[6] N. Patton, K. Rahmani, M. Missula, J. Biswas, and I. Dillig, "Programming-by-demonstration for long-horizon robot tasks," in *Proc. ACM Program. Lang.*, p. 34, vol. 8, Jan. 2024. [Online]. Available: https://doi.org/10.1145/3632860
[7] S. Orfanos and L. H. Lelis, "Synthesizing programmatic policies with actor-critic algorithms and relu networks," 2023, *arXiv:2308.02729*.
[8] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Using probabilistic movement primitives in robotics," *Auton. Robots*, vol. 42, pp. 529–551, 2018.
[9] R. M. Neal and G. E. Hinton, *A View of the EM Algorithm That Justifies Incremental, Sparse, and Other Variants*. Dordrecht, The Netherlands: Springer, 1998, pp. 355–368.
[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[11] A. Boularias, J. Kober, and J. Peters, "Relative entropy inverse reinforcement learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 182–189.
[12] S. Daftry, J. A. Bagnell, and M. Hebert, "Learning transferable policies for monocular reactive MAV control," in *Proc. Int. Symp. Exp. Robot.*, 2017, pp. 3–11.
[13] M. Bojarski et al., "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*.
[14] C. Chi et al., "Diffusion policy: Visuomotor policy learning via action diffusion," 2023, *arXiv:2303.04137*.
[15] P. Florence et al., "Implicit behavioral cloning," in *Proc. 5th Conf. Robot Learn.*, 2022, pp. 158–168.
[16] B. D. Ziebart et al., "Maximum entropy inverse reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2008, pp. 1433–1438.
[17] D. Brown, W. Goo, P. Nagarajan, and S. Niekum, "Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 783–792.
[18] C. Molnar, *Interpretable Machine Learning*. Morrisville, NC, USA: Lulu Press, 2020.
[19] J. P. Inala, O. Bastani, Z. Tavares, and A. Solar-Lezama, "Synthesizing programmatic policies that inductively generalize," in *Proc. Int. Conf. Learn. Representations*, 2019.
[20] D. Trivedi, J. Zhang, S.-H Sun, and J. J. Lim, "Learning to synthesize programs as interpretable and generalizable policies," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 25146–25163.
[21] Y. Li, J. Song, and S. Ermon, "InfoGAIL: Interpretable imitation learning from visual demonstrations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3815–3825.
[22] C. V. Perico, J. De Schutter, and E. Aertbeliën, "Learning robust manipulation tasks involving contact using trajectory parameterized probabilistic principal component analysis," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 8336–8343.
[23] J. Tremblay, T. To, A. Molchanov, S. Tyree, J. Kautz, and S. Birchfield, "Synthetically trained neural networks for learning human-readable plans from real-world demonstrations," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 5659–5666.
[24] F. Torabi, G. Warnell, and P. Stone, "Generative adversarial imitation from observation," 2018, *arXiv:1807.06158*.
[25] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, "Imitation from observation: Learning to imitate behaviors from raw video via context translation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 1118–1125.
[26] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 6325–6331.
[27] L. Zhou, C. Xu, and J. Corso, "Towards automatic learning of procedures from web instructional videos," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 7590–7598.
[28] I. Goodfellow et al., "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014.
[29] D. Blessing et al., "Information maximizing curriculum: A curriculum-based approach for training mixtures of experts," 2023, *arXiv:2303.15349*.
[30] J. Wei, J. Holtz, I. Dillig, and J. Biswas, "STEADY: Simultaneous state estimation and dynamics learning from indirect observations," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 6593–6599.
[31] J. P. Inala, O. Bastani, Z. Tavares, and A. Solar-Lezama, "Synthesizing programmatic policies that inductively generalize," in *Proc. Int. Conf. Learn. Representations*, 2020.
[32] A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson, "MPDM: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 1670–1677.
[33] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1, pp. 181–211, 1999.
[34] S. Thrun, "Probabilistic robotics," *Commun. ACM*, vol. 45, no. 3, pp. 52–57, 2002.
[35] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 5045–5054.
[36] A. Doucet et al., "A tutorial on particle filtering and smoothing: Fifteen years later," in *Handbook Nonlinear Filtering*, vol. 12, 2009, pp. 656–704.
[37] S. Gulwani et al., "Program synthesis," *Found. Trends Program. Lang.*, vol. 4, no. 1/2, pp. 1–119, 2017.
[38] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, no. 1–3, pp. 503–528, 1989.
[39] J. Holtz, A. Guha, and J. Biswas, "Interactive robot transition repair with SMT," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 4905–4911.
[40] E. Leurent, "An environment for autonomous driving decision-making," 2018. [Online]. Available: https://github.com/eleurent/highway-env
[41] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen, "panda-gym: Open-source goal-conditioned environments for robotic learning," in *Proc. 4th Robot Learn. Workshop: Self-Supervised Lifelong Learn.*, 2021.
[42] A. O. Ly and M. Akhloufi, "Learning to drive by imitation: An overview of deep behavior cloning methods," *IEEE Trans. Intell. Veh.*, vol. 6, no. 2, pp. 195–209, Jun. 2021.
[43] N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto, "Behavior transformers: Cloning k modes with one stone," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 22955–22968.
[44] S. Chaudhuri et al., "Neurosymbolic programming," *Found. Trends Program. Lang.*, vol. 7, no. 3, pp. 158–243, 2021.