One-Shot Learning of Autonomous Behaviour: A Meta Inverse Entailment approach

Dany Varghese[1], Daniel Cyrus[1*], Stassa Patsantzis[1] James Trewern[1], Alfie Anthony Treloar[2], Alan Hunter[2], and Alireza Tamaddoni-Nezhad[1]

[1] University of Surrey, Guildford GU1 7XH, UK
{dany.varghese,d.cyrus,s.patsantzis,a.tamaddoni-nezhad}@surrey.ac.uk
[2] University of Bath, Bath BA2 7AY, UK
{ajh210,aoat20}@bath.ac.uk

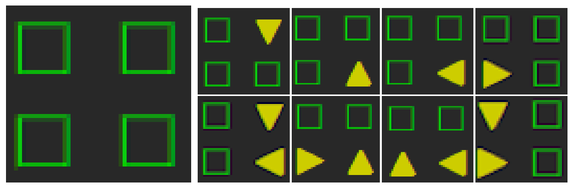
Abstract. "One-shot learning" traditionally refers to classifying a single instance using a machine learning model pre-trained on extensive datasets. In contrast, Meta Inverse Entailment (MIE), a type of Inductive Logic Programming (ILP), can generate complex logic programs from just a single positive example and minimal background knowledge without prior extensive training. This approach offers a human-centred form of machine learning that is more controllable, reliable, and comprehensible due to its small training data size and the inherent interpretability of logic programs. We use PyGol, a Python-based implementation of Meta Inverse Entailment, and compare its performance with ExpGen-PPO, a leading deep reinforcement learning system. Our experiments focus on two domains: maze-solving and obstacle avoidance for mobile robotics. In both domains, we first train the systems in simplified environments without obstacles and then test their ability to generalise to more complex environments with obstacles. Our results show that PyGol effectively learns generalisable solutions from a single example in both domains, whereas ExpGen-PPO requires more training and significantly more exploration to achieve similar performance.

Keywords: Meta Inverse Entailment · One-shot learning · Autonomous Systems · Reinforcement Learning · PyGol.

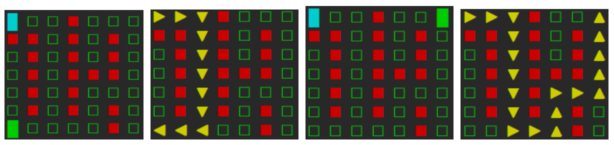
1 Introduction

Developing autonomous systems capable of rapidly learning and generalising complex behaviours from minimal data is a key challenge in artificial intelligence (AI) and robotics. Traditional machine learning methods often require large datasets and extensive training to achieve reliable performance in dynamic, real-world environments. In contrast, one-shot learning [5,19,17] aims to enable models to learn new tasks or behaviours from a single example, emulating the efficiency of human learning. This capability is crucial for applications where data is scarce, costly to obtain, or dynamically evolving, such as autonomous navigation, robotics, and intelligent decision-making systems.

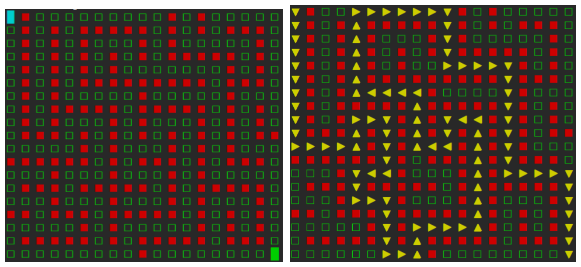
* Joint first author



(a) Training maze (b) Learned moves



(c) Testing mazes (7x7)



(d) Testing mazes (19x19)

Fig. 1: True One-shot learning of a maze-solving with PyGol.

(a) *PyGol is trained on a simplified maze with only floor tiles (green squares).*

(b) *PyGol learns a program whose transitions are all moves possible in any maze (yellow arrows).*

(c-d) *The learned program solves mazes of arbitrary size with walls, entrance, and exit tiles.*

This paper proposes a novel approach for one-shot learning of autonomous behaviour using Meta Inverse Entailment (MIE) [15,16,18], a symbolic machine learning technique that combines logic-based representations with inductive reasoning. By leveraging the inherent structure of logical rules and recursive predicates, MIE enables the automatic generation of compact, interpretable programs from minimal data. These programs can generalise learned behaviours to new, unseen environments while maintaining transparency and explainability.

We take advantage of a novel ILP approach called Meta Inverse Entailment, implemented within the PyGol framework [20], a Python-based system designed to support MIE's advanced learning capabilities. PyGol facilitates the automatic generation of language biases from background knowledge, eliminating the need for extensive predefined biases. By utilising bottom clauses of relevant literals and meta theory, PyGol enables the learning of target hypotheses in a flexible and adaptive manner, making it

One-shot maze-solving with PyGol
$solve(A, B) \leftarrow move\_down(A, B).$
$solve(A, B) \leftarrow move\_left(A, B).$
$solve(A, B) \leftarrow move\_right(A, B).$
$solve(A, B) \leftarrow move\_up(A, B).$
$solve(A, B) \leftarrow move\_down(A, C), solve(C, B).$
$solve(A, B) \leftarrow move\_left(A, C), solve(C, B).$
$solve(A, B) \leftarrow move\_right(A, C), solve(C, B).$
$solve(A, B) \leftarrow move\_up(A, C), solve(C, B).$

Table 1: The maze-solving program learned by PyGol from the training maze depicted in Figure 1a is shown here. The first four non-recursive clauses represent the agent's ability to exit the maze by taking a single step in each direction. The following four recursive clauses model continuous movement through the maze, enabling the agent to take multiple steps in one direction.

particularly effective in domains requiring one-shot learning. This framework enhances the generalisability of models, allowing them to solve complex tasks with minimal training data, as demonstrated in the autonomous behaviour experiments conducted in this study.

The one-shot generalisation capabilities of PyGol in a maze-solving domain are illustrated in Figure 1. PyGol is trained in a simplified environment without walls, containing only four empty floor tiles, as depicted in Figure 1b (green squares). Despite the simplicity of the training setup, PyGol learns all possible movement combinations in four directions, represented by yellow arrows in Figure 1b. The learned program can solve more complex mazes featuring walls, entrances, and exits, as shown in Figures 1c and 1d (with walls in red, entrances in blue, and exits in green), regardless of the maze's width or length. The underlying program, detailed in Table 1, demonstrates recursive calls to movement predicates: $move\_down/2$, $move\_up/2$, $move\_left/2$, and $move\_right/2$. These predicates enable the agent to take an arbitrary number of steps in any direction while ensuring that all moves are legal, i.e., the agent avoids crossing wall tiles. The move predicates are automatically generated based on the structure of the maze, highlighting PyGol's ability to efficiently generalise from minimal training data and solve a variety of maze configurations.

2 Related Work

One-shot learning is a machine learning paradigm that enables models to learn new tasks or recognise new objects from a single or very few examples, contrasting sharply with traditional statistical approaches that require large datasets for effective training. Conventional methods, such as deep neural networks, typically require extensive data to generalise accurately, often involving hundreds of thousands or even millions of labelled examples. This heavy reliance on large-scale datasets stems from the statistical nature of these models, which use high-volume data to capture patterns and reduce overfitting. As a result, traditional approaches can struggle in domains where data collection is costly or scarce, such as medical diagnosis, rare event detection, or personalised applications, making them less feasible for scenarios that demand rapid adaptation or when data is limited. The inability to generalise from minimal data not only hampers flexibility but also highlights a fundamental limitation of such models, prompting the exploration of alternative methods, like one-shot and few-shot learning, which are designed to address these challenges by leveraging prior knowledge and structured representations to learn efficiently from very few samples.

Recent efforts to bridge this gap include the concept of "in-context learning," introduced by Brown et al. (2020) with models like GPT-3 [2]. In-context learning enables large language models to perform tasks by conditioning on a few examples, but this is feasible only due to extensive pre-training on massive datasets involving billions of parameters. While this gives the appearance of one-shot or few-shot learning, the underlying reliance on large-scale pre-training raises concerns about scalability in truly resource-constrained settings. In contrast, one-shot learning techniques, such as Siamese networks [5], prototypical networks [12], and meta-learning algorithms like Model-Agnostic Meta-Learning (MAML) [4], focus on enabling models to generalise from minimal data. These methods often leverage prior knowledge and meta-level reasoning to enable more efficient learning from limited examples.

Meta-Interpretive Learning (MIL) has emerged as a significant approach in the field of one-shot learning by leveraging logical and symbolic representations to learn efficiently from minimal examples. One-shot learning within the ILP framework was first introduced by Muggleton [11], proving the feasibility of positive-only learning in a Bayesian context. In a more recent development, Muggleton [8] extended this work by proving error bounds for one-shot learning, assuming a target theory of minimal generality. This extension introduced DeepLog, a new implementation of MIL capable of performing positive-only, one-shot learning to generate minimal-size and correct hypotheses within the same Bayesian framework.

3 Meta Inverse Entailment & PyGol

Traditional Meta-Interpretive Learning approaches rely heavily on metarules, which serve as a declarative language bias to guide the hypothesis search. While this bias is essential for constraining and directing the learning process, it often demands significant human expertise and input, particularly in crafting domain-specific rules. This reliance on manually defined biases presents a challenge, as it limits the scalability and adaptability of MIL in various applications. This study introduces a novel ILP approach called Meta Inverse Entailment to address these limitations.

MIE circumvents the need for predefined declarative language biases by employing two key mechanisms: the Bottom Clause of Relevant Literals (BCRL) and Meta Theory (MT). BCRL efficiently gathers all literals relevant to an example based on background knowledge, establishing a well-defined boundary for hypothesis search during the learning process. This allows the model to explore a constrained yet comprehensive hypothesis space. Meta Theory, on the other hand, represents a higher-order language bias that is automatically derived from background knowledge, thus reducing the dependency on human-designed biases. Together, BCRL and MT enable a more flexible and autonomous hypothesis generation process, improving the model's ability to generalise across domains without significant manual intervention. While a full explanation of MIE's mechanisms is beyond the scope of this paper, the key concepts outlined here demonstrate its potential to overcome the traditional limitations associated with language biases in MIL.

Definition 1 (Relevant literals ($\overrightarrow{\mathcal{R}_e}$)). *Let B be the background knowledge, $\overrightarrow{\mathcal{B}}$ be the ordered clause of ground literals of B, and e be an example. Then the relevant literals of e is defined as $\overrightarrow{\mathcal{R}_e} = e \leftarrow L_1, L_2, \cdots, L_n$ if and only if $L_i \in \overrightarrow{\mathcal{B}}$ and L_i be related literal (See Definition 3 in [16]) of either L_j or e such that $1 \leq j < i$.*

Definition 2 (Bottom clause of relevant literals ($\bot_{e,B}$)). *Let B be the background knowledge, e be a definite clause representing an example, $\mathcal{K}$ be a set of constant terms and $\overrightarrow{\mathcal{R}_e}$ be the relevant literals of example e as defined in the Definition 1. Then bottom clause of relevant literals of e, denoted as $\bot_{e,B}$ is $\overrightarrow{\mathcal{R}_e}$ where each unique occurrence of term $t_i \notin \mathcal{K}$ replaced with a new variable.*

The bottom clause of relevant literals introduced in MIE can resemble the bottom clause concept in Progol [9], but it does not use language bias such as mode declaration.

Definition 3 (Meta theory of relevant literals ($\mathcal{M}_e$)). *Let e be an example, $\overrightarrow{\mathcal{R}_e}$ be the relevant literals of e. Then, a clause $\overrightarrow{C} \in \mathcal{M}_e$ if and only if;*

1. *$\exists \Theta$ such that $\overrightarrow{C}\Theta \succeq_s \overrightarrow{\mathcal{R}_e}$ (Refer Definition 19 in [13]) and*
2. *$\overrightarrow{C}$ is head_connected (Refer Definition 3 in [10]).*

The significant difference between a meta clause and a metarule is that a metarule specifies a relationship between the variables in the body of a rule. In meta theory, however, we avoid the tightly coupled nature of metarules to reduce computational complexity, deriving directly from background knowledge.

In MIE, we introduce a novel concept called the double-bounded hypothesis set. This set is defined as a sequential subsumption of meta theory, relative to the bottom clause of related literals. In this new framework, the hypothesis set of an example is constrained within specific boundaries: the bottom clause of relevant literals at the lower boundary and the encompassing meta theory at the upper boundary.

Definition 4 (Double-bounded hypothesis set ($HS(\bot_{e,B}, \mathcal{M}_e)$)). *Let e be an example, $\bot_{e,B}$ be a bottom clause of relevant literals of e as defined in Definition 2, $\mathcal{M}_e$ be a meta theory as defined in Definition 3. Then*

$$HS(\bot_{e,B}, \mathcal{M}_e) = \{h| \text{ s.t: (1) } h \in \overrightarrow{\mathcal{L}_\bot}, \text{ and (2) } h \in \overrightarrow{\mathcal{L}_\mathcal{M}}\}$$

Now, Consider the problem specification of ILP. Suppose we are given a Horn program for a background knowledge B and a single Horn clause as an example E, then the hypothesis H should follow;

$$B \wedge H \models E \tag{1}$$

$$B \wedge H \textit{ is consistent} \tag{2}$$

Equations 1 & 2 are equivalent to

$$B \wedge \neg E \models \neg H \tag{3}$$

$$B \not\models \neg H \tag{4}$$

Like inverse entailment, MIE generates some theories that are not derived from B alone but from $B \wedge \neg E$. Instead of the bottom clause, we consider a bridge formula CT. So Equation 3 can be written as

$$B \wedge \neg E \models CT \tag{5}$$

$$CT \models \neg H \tag{6}$$

Finally, Equation 6 can also written as

$$H \models \neg CT \tag{7}$$

Also by Equations 4 and 6, we have

$$B \not\models CT \tag{8}$$

By Equation 5, the set of clausal theories, CT, can be obtained by generating the double-bounded hypothesis set. Then

$$HS(\bot_{e,B}, \mathcal{M}_e) \models \neg CT \tag{9}$$

The candidate hypothesis (H) will be a subset of $HS(\bot_{e,B}, \mathcal{M}_e)$.

$$H = \{h | h \in HS(\bot_{e,B}, \mathcal{M}_e) \text{ and } h \models \neg CT\} \tag{10}$$

In Equation 5, we introduce a bridge formula, a set of clausal theories as CT, which replaces the concept of the bottom clause.

4 Experiments

We explore the potential of positive-only learning across two experimental settings where negative examples are unnecessary: (a) path planning in a maze, and (b) in a free-form robotics simulation[3]. In both environments, the physical constraints make negative examples redundant, as they are captured by the First-Order background theory provided by the user. Specifically, the agent is allowed to take actions that do not result in crossing obstructions—walls in the maze and obstacles in the robotics simulation—ensuring that all movements are valid according to the physics of the environment.

[3] The code for this study is available on GitHub at `https://github.com/hmlr-lab/One-Shot-Learning-autonomous-behavior.git`.

In these settings, to learn how to navigate around obstacles, the agent only needs to master how to move between any two points while exploring all possible valid paths. This effectively demonstrates the agent's ability to generalise based solely on positive examples. If the physical laws governing the environment are not explicitly provided as background knowledge, negative examples, such as attempts to walk through a wall, would be required to define the boundaries of valid actions. Investigating how generalisation occurs with the presence of negative examples is an area left for future study.

4.1 Maze solving

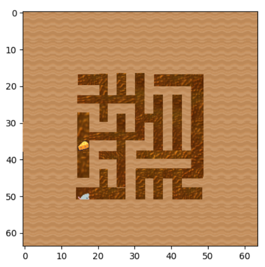
Mazes play a crucial role in the study of planning and navigation within artificial intelligence and robotics [7]. They provide a controlled and structured environment for testing algorithms that involve pathfinding, decision-making, and obstacle avoidance. Mazes challenge agents to find the most efficient route from a start point to a goal while navigating through various barriers and dead ends, making them ideal for evaluating an agent's ability to plan, learn, and adapt in complex environments [14]. The simplicity of a maze's design, combined with its potential complexity, allows researchers to study fundamental principles of planning and movement strategies that are applicable to real-world scenarios such as robotics, autonomous vehicles, and exploratory missions.

PyGol setup In our experiments, we first train the model within a simplified environment: an empty maze consisting of four passable "floor" tiles with no internal walls, as visualised in Figure 1a. This minimal setup is designed to reduce complexity and allow the model to focus on basic movement operations. We define four *move actions* — $move\_down/2$, $move\_up/2$, $move\_left/2$, and $move\_right/2$ — which represent the agent's ability to move one step in each of the cardinal directions.

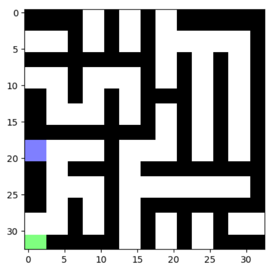
PyGol uses these inputs to autonomously learn a generalised program for navigating the maze. The goal is for PyGol to identify all valid movement combinations and generalise a navigation strategy that can be applied in more complex mazes. During training, the model learns the movement operations as logical rules, which are then compiled into a formalised program, as shown in Table 1.

By training in this controlled, empty maze environment, the model efficiently acquires the foundational rules necessary for path planning. These learned rules can later be transferred to more complex domains involving obstacles, walls, and varying maze sizes, enabling the agent to navigate and solve more intricate problems. This experimental setup demonstrates the ability of PyGol to generalise from minimal examples while building a robust movement strategy applicable across different scenarios.

ExpGen-PPO setup For the Maze Solving experiment, we use the Maze environment from ProcGen Benchmark [3], a set of 16 computer game-like environments designed to test the “Zero-shot generalisation" ability of Reinforcement Learning (RL) systems. Each environment is divided into procedurally generated “levels" randomly assigned to training and testing sets. In the Maze environment, each level is one maze, with dimensions varying from 3×3 to 25×25, and with walls placed by Kruskal's algorithm [6]. Each maze has a start and end location, placed at random and an agent can navigate a maze by taking one of four discrete *move actions*: *moving up, down, left* or



(a)



(b)

Fig. 2: ProcGen maze environments :(2a) Original Environment (2b) Modified Environment.

right. We modified the ProcGen environment[4], as it originally generates random mazes each time with a fixed total environment size. To make it compatible with the ILP approach, adjustments were made to ensure consistency and structure that aligns with ILP requirements. Figure 2 shows the original and modified ProcGen environments.

For the testing phase, we evaluated the performance of the two systems, PyGol and ExpGen-PPO, on a set of 15 mazes divided into three distinct sizes: 5×5, 10×10, and 15×15. Each size category contained five unique mazes, providing a comprehensive assessment of the systems' ability to generalise across varying complexity levels. The primary metric used for evaluation was the number of steps each system required to locate the exit in each test maze. This metric includes both forward steps and any backtracking actions taken during navigation.

The results of these tests are illustrated in Figure 3, where the average number of steps is plotted for each system across the different maze sizes. Error bars are included to represent the standard error, providing insight into the variability of the performance across the test mazes. This evaluation allows us to compare the efficiency and consistency of both systems when solving mazes of increasing complexity. We also reported the number of steps executed by a Prolog-based solver[5] that employs backtracking to find paths using the program generated by PyGol.

4.2 Obstacle avoidance

We conducted a comparative analysis of PyGol and ExpGen-PPO for pathfinding and obstacle avoidance within a robotic simulation environment[6]. This simulator, developed

[4] The modified version of ProcGen for fixed maze size is available at `https://github.com/hmlr-lab/procgen`.

[5] The code is available at `https://github.com/JamesTrewern/louise-Testing/tree/master/data`.

[6] The code for the simulator is available at `https://github.com/aoat20/survey-simulation`.

Episode	Map	Variants	Islands	Distance
1 (training)	River	1	0	44.72
2 (testing)	River	6	1	210.95
3 (testing)	River	6	1	63.25
4 (testing)	Lake	6	2 or 3	174.93

Table 2: Obstacle avoidance training and test instances

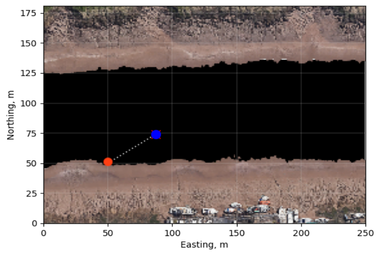
in Python, models a survey mission in which an uncrewed surface vessel (USV) scans the bottom of a body of water using sonar to locate an object of interest. The simulation environment is depicted in Figure 3a, which presents an example of a complete simulation run (or "episode") used during our experiments.

In this scenario, the USV operates in a river environment, where the dark central area represents the water, and the coloured regions above and below depict the riverbanks, which are impassable for the vessel. The USV itself is represented as a blue dot, while two red patches located to the south-west of the USV mark the areas that have been scanned by its sonar sensors. A narrow, unscanned strip, known as the "nadir," runs between the sonar's coverage areas. Users can interact with the simulator by clicking on a location in the window, directing the USV to travel to the selected coordinates. In the figure, the user has clicked on a point to the northeast of the USV's current position, and the simulation captures the vessel's movement towards that destination, as well as the regions that have already been scanned. This setup provides a controlled environment to test the pathfinding capabilities of the USV, including its ability to avoid obstacles while completing its mission.

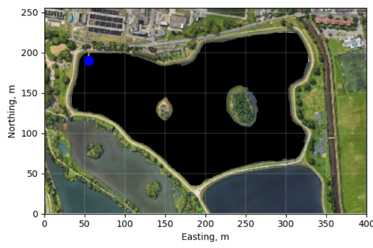
PyGol setup In this experiment, similar to the maze-solving scenario, we manually define movement actions for the uncrewed surface vessel (USV). Instead of the four basic directional actions, we expand the agent's capability by introducing eight dyadic actions, enabling it to move in all eight compass directions: $North$, $North-East$, $East$, $South-East$, $South$, $South-West$, $West$, and $North-West$. These expanded movement options allow for more flexible and precise navigation through the environment, particularly in avoiding obstacles and efficiently reaching the destination.

To complement the agent's movement capabilities, we have developed a Breadth-First Search (BFS) algorithm to find the shortest path in the given environment. The BFS algorithm systematically explores all possible paths from the USV's starting position, ensuring that the shortest and most optimal route is identified. This approach is particularly effective in environments where obstacles such as islands or other impassable areas are present, as it guarantees that the agent will always take the most efficient path to avoid these barriers while reaching the target destination. The use of BFS in this context not only ensures path efficiency but also improves overall performance in scenarios involving complex navigation tasks.

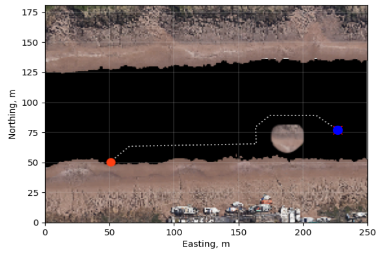
In this approach, we encountered challenges using BFS during the training phase, as it proved difficult to capture all possible action moves within a single optimal path. Typically, in training episodes without obstacles, the shortest path tends to be a direct line to the goal, which limits exposure to various action moves as shown in Figure



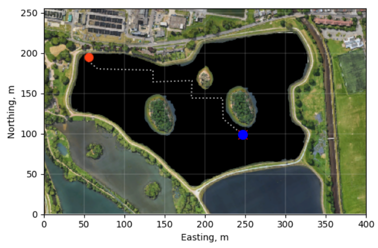
(a)



(b)



(c)



(d)

Fig. 3: Obstacle avoidance environments. 3a: Training episode (Episode 1). 3b: Lake map. 3c, 3d: Paths learned by PyGol. White dots indicate the moved path, the red dot marking the starting position and the blue dot marking the ending position.

3a. To address this, we introduce the concept of *K-path* learning during training. This approach aims to learn *K distinct paths that are near-optimal*. By diversifying the paths explored, K-path learning ensures that the resulting learned program encompasses the full range of action moves. Table 3 presents the programs generated when varying the value of K. The inclusion of multiple paths improves the completeness and robustness of the final rules, incorporating all necessary action moves for a well-rounded learned model.

To enhance realism, we also introduce additional constraints for each movement action. The first is a passability constraint, which prevents the USV from moving to impassable areas, such as land or riverbanks, as illustrated in Figure 3a. The second is a directionality constraint, which ensures that the USV only takes actions that decrease the Euclidean distance between its current location and the target destination.

We create four simulation episodes—one for training and three for testing—each with a different initial position and destination. Three episodes occur in a river environment and one in a lake environment, as shown in Figure 3. For each map, we manually introduce one or two islands to obstruct the USV's path. The lake map already contains two islands, so we added a third to increase the complexity. This results in one training

$move(A, B) : -move\_east(A, B).$
$move(A, B) : -move\_east(A, C), move(C, B).$
$move(A, B) : -move\_south\_east(A, B).$
$move(A, B) : -move\_south\_east(A, C), move(C, B).$

(a) PyGol Program (K=1)

$move(A, B) : -move\_east(A, B).$
$move(A, B) : -move\_east(A, C), move(C, B).$
$move(A, B) : -move\_south\_east(A, B).$
$move(A, B) : -move\_south\_east(A, C), move(C, B).$
$move(A, B) : -move\_west(A, B).$
$move(A, B) : -move\_west(A, C), move(C, B).$
$move(A, B) : -move\_south(A, B).$
$move(A, B) : -move\_south(A, C), move(C, B).$

(b) PyGol Program (K=2)

Table 3: Program learned by PyGol for different values of K

scenario and 18 testing variations, differing in the distance to the destination and the number and position of islands.

The USV cannot navigate through islands, so the agent must find an alternative path. Table 2 outlines the training and testing cases, where the "Distance" column lists the Euclidean distance between the initial and destination coordinates. To train PyGol, we use a single training example—a ground atom of the dyadic predicate $move/2$—with input and output state derived from the initial and final positions in Episode 1 from Table 1.

ExpGen-PPO setup We train ExpGen-PPO using the OpenAI Gym toolkit for RL research [1]. The river map image, without any islands, is transformed into a binary matrix and incorporated into a Gym setting. The ExpGen-PPO training function creates 100 random destinations, equivalent to 100 training instances. We set the agent's reward to 10 for reaching the coordinates of an instance's destination, -1 for moving on impassable terrain, 1 for moving towards a training instance's destination and 0 for moving away from the destination. At test time, we set the starting position and destination to each of the 18 test instances and transform the lake image into a binary matrix for the 6 testing instances in the lake map.

We test PyGol and ExpGen-PPO on each of our 18 test instances. We measure the number of steps taken to find a path from the starting position to the destination of each episode, including backtracking steps. Figure 4b illustrates the results. The error bars show standard error.

5 Results & Discussion

The results, as illustrated in the figures 4a and 4b, compare the performance of PyGol and ExpGen-PPO across different maze sizes and simulation episodes. We additionally

$move(A, B) : -move\_east(A, B).$
$move(A, B) : -move\_east(A, C), move(C, B).$
$move(A, B) : -move\_south\_east(A, B).$
$move(A, B) : -move\_south\_east(A, C), move(C, B).$
$move(A, B) : -move\_west(A, B).$
$move(A, B) : -move\_west(A, C), move(C, B).$
$move(A, B) : -move\_north\_west(A, B).$
$move(A, B) : -move\_north\_west(A, C), move(C, B).$
$move(A, B) : -move\_south(A, B).$
$move(A, B) : -move\_south(A, C), move(C, B).$
$move(A, B) : -move\_south\_west(A, B).$
$move(A, B) : -move\_south\_west(A, C), move(C, B).$
$move(A, B) : -move\_north(A, B).$
$move(A, B) : -move\_north(A, C), move(C, B).$
$move(A, B) : -move\_north\_east(A, B).$
$move(A, B) : -move\_north\_east(A, C), move(C, B).$

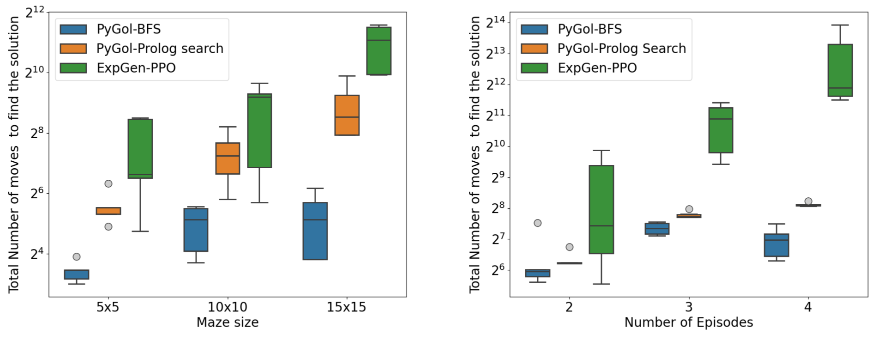
Table 4: The final program learned by PyGol from obstacle avoidance (K=4)

recorded the step count of a Prolog-based solver, which utilises backtracking to identify paths based on the program developed by PyGol. In Figure 4a, representing the maze-solving experiment, PyGol demonstrates superior efficiency across all maze sizes, with significantly fewer moves required to solve the maze than ExpGen-PPO. As the maze size increases, the performance of ExpGen-PPO worsens, with a large increase in the number of moves, while PyGol remains relatively consistent, particularly in the 10×10 and 15×15 mazes. Moreover, the Prolog solver was able to solve the mazes in a relatively lower number of steps compared to ExpGen-PPO.

In Figure 4b, which shows performance in the obstacle avoidance simulation, the trend continues. PyGol consistently outperforms ExpGen-PPO in all episodes, especially in more complex environments (Episode 3 and Episode 4). The number of moves required for ExpGen-PPO escalates with the complexity of the environment, whereas PyGol maintains (using BFS and Prolog solver)a low and stable move count, indicating its effectiveness in handling both simpler and more challenging map scenarios.

PyGol, from a single example, learns a recursive program capable of implementing a general path-finding strategy. This learned program enables the agent to navigate to any destination east of its starting position, as demonstrated in the training episode (Episode 1). By combining this general path-finding capability with the passability constraints provided by the user, PyGol efficiently guides the robot around obstacles to reach its destinations. The learned program for obstacle avoidance is detailed in Table 4.

Both PyGol and ExpGen-PPO successfully solved all test instances in both the obstacle avoidance and maze-solving domains. However, a key distinction lies in their approaches when encountering unfamiliar environments. ExpGen-PPO, when its trained policy proves insufficient, resorts to a costly exploration-based strategy, significantly increasing the time required to solve most test instances. In contrast, PyGol, with its learned recursive path-finding program, efficiently navigates unseen environments.



(a) (b)

Fig. 4: Maze Solving (4a) and Obstacle Avoidance (4b) performance.

The maze-solving results further emphasise this difference. As the maze size increased, PyGol consistently outperformed ExpGen-PPO by requiring significantly fewer moves to reach the goal. ExpGen-PPO's reliance on exploration led to a considerable increase in the number of moves, particularly in larger and more complex mazes. This raises questions about the generalisability and robustness of policies trained via reinforcement learning methods like ExpGen-PPO, especially when applied to environments beyond those encountered during training. PyGol's ability to generalise from minimal training examples suggests a more scalable and adaptable solution for both obstacle avoidance and maze-solving tasks in dynamic and novel environments.

6 Conclusions

In conclusion, this study has demonstrated the efficacy of Meta Inverse Entailment (MIL) as a powerful framework for learning generalised behaviours from minimal data. Through both theoretical validation and empirical evaluation, we introduced and evaluated PyGol, a novel MIE-based system, compared to the state-of-the-art Deep Reinforcement Learning (Deep-RL) approach, ExpGen-PPO. We applied these systems to two path-finding tasks: one in a simplified maze environment and another in a robotic simulation. Our results reveal that PyGol can efficiently learn complex behaviours from a single training example in simplified environments, such as an empty maze or a river simulation, and successfully generalise this knowledge to more complex test environments, including mazes with random dimensions and walls, as well as river and lake simulations containing obstacles like islands.

Although ExpGen-PPO demonstrated the ability to generalise behaviour, it required more extensive and task-specific training, such as exposure to additional maze levels or randomly varied destinations, to achieve similar performance to PyGol. The key advantage of PyGol lies in its capacity to generalise robustly from minimal input, reducing the need for exhaustive training across diverse scenarios.

These findings underscore the potential of MIE for applications where efficient learning from sparse data is critical. PyGol's performance highlights the adaptability and scalability of ILP approaches for solving complex tasks, making it a valuable tool for domains that involve dynamic, novel, or resource-constrained environments. This work contributes to advancing the field of machine learning by offering a more efficient alternative to data-intensive methods for path-finding and similar tasks.

Acknowledgement

The authors acknowledge the EPSRC grant on Human-machine learning of ambiguities to support safe, effective, and legal decision making (EP/X030156/1). Alireza Tamaddoni-Nezhad also acknowledges the EPSRC Network Plus grant on Human-Like Computing (HLC).

References

1. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)
2. Brown, T., Mann, B., et. al., R.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020)
3. Cobbe, K., Hesse, C., Hilton, J., Schulman, J.: Leveraging procedural generation to benchmark Reinforcement Learning. arXiv preprint arXiv:1912.01588 (2019)
4. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70. p. 1126–1135. ICML'17, JMLR.org (2017)
5. Koch, G., Zemel, R., Salakhutdinov, R., et al.: Siamese neural networks for one-shot image recognition. In: ICML deep learning workshop. vol. 2:1, pp. 1–30. Lille (2015)
6. Kruskal, J.B.: On the shortest spanning subtree of a graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society **7**, 48–50 (1956)
7. LaValle, S.M.: Planning algorithms. Cambridge university press (2006)
8. Muggleton, S.H.: Hypothesizing an algorithm from one example: the role of specificity. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **381**(2251), 20220046 (2023)
9. Muggleton, S.: Inverse entailment and Progol. New Generation Computing **13**(3), 245–286 (Dec 1995)
10. Muggleton, S., Tamaddoni-Nezhad, A.: Qg/ga: A stochastic search for progol. In: Inductive Logic Programming: 16th International Conference, ILP 2006, Santiago de Compostela, Spain, August 24-27, 2006, Revised Selected Papers 16. pp. 37–39. Springer (2007)
11. Muggleton, S.H.: Learning from positive data. In: Inductive Logic Programming Workshop (1996), https://api.semanticscholar.org/CorpusID:18451163
12. Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper_files/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf
13. Tamaddoni-Nezhad, A., Muggleton, S.: The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. Machine learning **76**, 37–72 (2009)

14. Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics (massachusetts (2005)
15. Varghese, D.: Explainable and Efficient Machine Learning using Meta Inverse Entailment. Ph.D. thesis, University of Surrey (2024). https://doi.org/10.15126/thesis.901077
16. Varghese, D., Barroso-Bergada, D., Bohan, D.A., Tamaddoni-Nezhad, A.: Efficient Abductive Learning of Microbial Interactions using Meta Inverse Entailment. In Proceedings of the 31st International Conference on ILP (2022), (in press)
17. Varghese, D., Bauer, R., Baxter-Beard, D., Muggleton, S., Tamaddoni-Nezhad, A.: Human-like rule learning from images using one-shot hypothesis derivation. In: Inductive Logic Programming. pp. 234–250. Springer International Publishing, Cham (2022), https://link.springer.com/chapter/10.1007/978-3-030-97454-1_17
18. Varghese, D., Patel, U., Krause, P., Tamaddoni-Nezhad, A.: Few-shot learning for plant disease classification using ilp. In: Garg, D., Narayana, V.A., Suganthan, P.N., Anguera, J., Koppula, V.K., Gupta, S.K. (eds.) Advanced Computing. pp. 321–336. Springer Nature Switzerland, Cham (2023)
19. Varghese, D., Tamaddoni-Nezhad, A.: One-shot rule learning for challenging character recognition. In Proceedings of the 14th International Rule Challenge, Oslo, Norway **2644**, 10–27 (08 2020), http://ceur-ws.org/Vol-2644/paper44.pdf
20. Varghese, D., Tamaddoni-Nezhad, A.: PyGol. https://github.com/PyGol/ (2022)