Solving Minecraft Tasks via Model Learning

Yarin Benyamin, Argaman Mordoch, Shahaf Shperberg, Roni Stern

Ben-Gurion University of the Negev
bnyamin@post.bgu.ac.il, mordocha@post.bgu.ac.il, shperbsh@bgu.ac.il, roni.stern@gmail.com

Abstract

Minecraft is a sandbox game that offers a rich and complex environment for AI research. Its design allows for defining diverse tasks and challenges for AI agents, such as gathering resources and crafting items. Previous works have applied both Reinforcement Learning (RL) and Automated Planning methods to accomplish different tasks in Minecraft. RL methods usually require a large number of interactions with the environment, while planning methods require a model of the domain to be available. Creating planning domain models for Minecraft tasks is arduous. Algorithms for learning a planning domain model from observations exist, yet they have mostly been used on planning benchmarks. In this work, we explore the use of such algorithms for solving Minecraft tasks. We propose an agent that learns domain models from observations—either generated by an expert or collected online—and uses them with an off-the-shelf domain-independent planner. As a case study, we explore how such an agent can be used for the task of crafting a wooden pogo stick. Experimental results demonstrate the benefit of domain model learning and planning over standard RL-based methods.

Introduction

Minecraft is a widely popular sandbox game that offers a rich and complex environment for AI research. Its design allows for defining different tasks for Artificial Intelligence (AI) agents to perform, such as gathering resources and crafting items. Building AI agents that can play Minecraft and accomplish such tasks has been acknowledged as a major AI challenge and has received significant interest in the academic community. This includes an annual competition in NeurIPS (Guss et al. 2019a) and a dedicated game mod and framework for evaluating AI agents (Goss et al. 2023).

The state-of-the-art approach to building AI agents for solving Minecraft tasks is by applying RL methods (Tessler et al. 2017; Frazier and Riedl 2019; Scheller, Schraner, and Vogel 2020). When using these methods, the AI agent learns to make decisions by interacting with an environment and receiving feedback through states and rewards (or penalties). Utilizing RL methods comes with its drawbacks. The learning phase demands substantial computational resources, prolonged training periods, and extensive interactions with the environment. The latter downside becomes particularly pronounced when developing agents for video games, such as Minecraft, in which environment interaction is computationally expensive and often intentionally slowed down to accommodate human gameplay.

Automated planning methods have also been used to build AI agents for Minecraft (Roberts et al. 2017; Wichlacz, Torralba, and Hoffmann 2019) and video games in general (Duarte et al. 2020; Bartheye and Jacopin 2009). One of the advantages of using planning is that it does not require interacting with the environment to decide which actions to perform to achieve desired goals. Additionally, due to the symbolic nature of most planning algorithms, the resulting plans for the planning tasks are more explainable than the policies generated by RL agents (Hoffmann and Magazzeni 2019). A significant limitation of automated planning lies in its dependency on a domain model. Modeling a domain can be very difficult, even for human experts. Prior work proposed automated methods for learning from observations state representations (Konidaris, Kaelbling, and Lozano-Perez 2018) and action models (Juba, Le, and Stern 2021; Wang 1994; Aineto, Celorrio, and Onaindia 2019). While action model learning methods have shown some promise, their application beyond standard automated planning benchmarks remains limited.

This work explores how automated planning can be used to build a Minecraft-playing agent, eliminating the need for a human modeler to manually provide an action model. Specifically, we propose a Minecraft-playing agent called PDDL-agent, which uses Numeric Safe Action Model Learning (N-SAM) (Mordoch, Juba, and Stern 2023), a state-of-the-art action model learning algorithm, to learn an action model from observations. Our PDDL-agent passes the learned domain model to an off-the-shelf domain-independent numeric planner, namely Metric-FF (Hoffmann 2003), to select which actions it should perform. Our PDDL-agent is designed to work in two settings: offline and online. In the *offline* setting, it receives observations of an expert acting in the domain, and then it has to generate a plan to perform some Minecraft task. In the *online* setting, the agent is not given any observations a priori and must determine how to interact with the environment to acquire the necessary knowledge for solving the desired Minecraft task. We propose a novel hybrid strategy that integrates RL and automated planning algorithms. In this hybrid strategy, the agent first attempts to construct a plan based on its current

learned action model. If successful, it executes the generated plan in the environment. If generating a plan proves infeasible under the current action model, an RL algorithm is employed to engage with the environment. This combination of RL and planning yields mutual benefits. Planning leverages RL as a methodological means to explore the environment and gather observations, achieving a balance between exploration and goal-oriented exploitation. Simultaneously, RL is trained using the trajectories executed via the plan, thereby enhancing its problem-solving capabilities.

We evaluated our PDDL-agent in Polycraft, a symbolic wrapper to Minecraft, for the task of crafting a wooden pogo stick. Compare its performance with appropriate RL-based agents. We considered two symbolic representations of the domain and evaluated our agents in both the offline and online settings. In the offline setting, our PDDL-agent dominates the RL baselines in most cases and demonstrates impressive zero-shot transfer capabilities. In the online setting, our hybrid strategy successfully harnesses action models learning and planning even when expert trajectories are not provided, outperforming PPO, a well-established RL algorithm.

Background and Related Work

Planning problems in domains where action outcomes are deterministic, states are fully observable, and the states are described with discrete and continuous state variables, can be defined using a subset of the Planning Domain Definition Language (PDDL) (Aeronautiques et al. 1998).[1] In PDDL, a planning domain is defined by a tuple $D = \langle F, X, A \rangle$ where F is a finite set of Boolean variables, X is a set of numeric variables, and A is a set of actions. A state is an assignment of values to all variables in $F \cup X$. Every action $a \in A$ is defined by a tuple $\langle \mathit{name}(a), \mathit{pre}(a), \mathit{eff}(a) \rangle$ representing the action's name, preconditions, and effects, respectively. Preconditions are assignments over the Boolean variables and conditions over the numeric variables, specifying when the action can be applied. The effects of action are a set of assignments over F and X, representing how the state changes after applying a. The set of actions with their definitions is referred to as the *action model* of the domain. A planning problem in PDDL is defined by $\langle D, s_0, G \rangle$ where D is a domain, s_0 is the initial state, and G are the problem goals. The problem goals G are assignments of values to a subset of the Boolean variables and a set of conditions over the numeric variables. A solution to a planning problem is a *plan*, i.e., a sequence of actions applicable in s_0 and resulting in a state s_G in which G is satisfied.

Algorithms for learning action models from observations vary in the assumptions they make on the available observations and the guarantees they provide on the action model they return (Cresswell, McCluskey, and West 2013; Amir and Chang 2008; Yang, Wu, and Jiang 2007; Aineto, Celorrio, and Onaindia 2019; Juba, Le, and Stern 2021). For example, FAMA (Aineto, Celorrio, and Onaindia 2019) can handle missing observations while SAM Learning (Stern and Juba 2017; Juba, Le, and Stern 2021) cannot. LOCM (Cresswell, McCluskey, and West 2013) can even learn an action model only from observed sequences of actions. On the other hand, FAMA and LOCM only guarantee that the learned action model is consistent with the given observations, while SAM Learning guarantees any plan generated with the learned action model is valid w.r.t. the real, unknown, action model. PlanMiner (Segura-Muros, Pérez, and Fernández-Olivares 2021) and the Numeric Safe Action Model Learning (N-SAM) algorithms (Mordoch, Juba, and Stern 2023) are, to the best of our knowledge, the only algorithms capable of learning action models that include numeric preconditions and effects. N-SAM possesses several appealing properties. First, it runs in polynomial time w.r.t. the input data. Second, it guarantees that every plan generated with it is valid w.r.t. the real, unknown, domain model. The PDDL agent described in this work relies on N-SAM.[2]

Reinforcement Learning Algorithms

RL is a field of AI in which agents learn to make decisions by interacting with an environment and receiving feedback in the form of rewards (or penalties). In this work, we focus on two key RL algorithms, Deep Q-Network (DQN) (Mnih et al. 2015) and Proximal Policy Optimization (PPO) (Schulman et al. 2017) These algorithms hold a prominent status within the RL community. DQN is an off-policy RL algorithm implementation of Q-Learning (Watkins 1989) that uses deep neural networks to solve problems in large state spaces. PPO is an on-policy RL algorithm that alternates between sampling data through interaction with the environment and optimizing a "surrogate" objective function using stochastic gradient descent. Notably, PPO often outperforms alternative RL algorithms across various domains and is renowned for its robust performance even in the absence of extensive hyperparameter tuning. An interesting distinction lies in the fact that off-policy algorithms, such as DQN, can learn from a given set of observations (comprising state, action, next state, and reward), while on-policy algorithms, like PPO, rely on directly utilizing data collected during interaction with the environment to update their policy.

Imitation Learning (IL) (Pomerleau 1991) is a related AI field where an AI agent is trained to "mimic human behavior in a given task" (Hussein et al. 2017). Two IL algorithms we discuss in this work are Behavioral Cloning (BC) (Bratko, Urbančič, and Sammut 1995) and Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon 2016). BC employs supervised learning to mimic the expert's policy, while GAIL takes a unique adversarial approach by simultaneously training a policy and a discriminator. The discriminator's role is to distinguish between expert observations and those generated by the learned policy. Offline RL (Kumar et al. 2020) is similar to IL except that the given trajectories encompass not only the states and actions but also the rewards associated with each transition. Offline RL algorithms aim not to mimic these trajectories but to learn from them how to maximize future rewards. Consequently, off-policy algo-

[1] Since our problem includes discrete and continuous state variables, we require PDDL2.1 (Fox and Long 2003).

[2] Technically speaking, our PDDL agent uses N-SAM* (Mordoch et al. 2023), a recently proposed advanced version of N-SAM.

rithms, such as DQN, can also serve as an offline RL algorithm.

Minecraft Environments

MineRL (Guss et al. 2019b) serves as an OpenAI-Gym (Brockman et al. 2016) compatible research environment, providing a Minecraft-based platform for the development, testing, and evaluation of RL algorithms. However, it is not suitable for our work as we do not consider a visual, pixel-based representation of the game. Instead, we use Polycraft (Palucka 2017), an interface to Minecraft that is part of the Polycraft World AI Lab(PAL) (Goss et al. 2023)[3]. PAL allows AI agents to easily interact with Minecraft's environment by sending commands to the API and waiting for a response. Each command has pre-defined preconditions, effects, and costs. This mechanism enables RL algorithms to use the API to train their agents and easily solve various tasks. Unlike MineRL, PAL supports *symbolic* observations, ideal for planning algorithms, which require a symbolic model of the environment to solve problems.

AI Agents for Minecraft

Planning and RL have been used to design AI agents for Minecraft. Wichlacz et al. (2019) used PDDL modeling to solve complex construction tasks in Minecraft. They modeled house-construction tasks as classical and as Hierarchical Task Network (HTN) (Georgievski and Aiello 2015) planning problems. They observed that even simple tasks present difficulties to current planners as the size of the world increases. The HTN planner scaled well when the size of the world increased but was too coupled with the specific task. Learning HTN domains from observations is an open problem.

The most widely adopted method for playing Minecraft is the Hierarchical Deep Reinforcement Learning Network (H-DRLN) (Tessler et al. 2017). This approach enables the agent to continuously learn multiple policies and adapt to new challenges within the game. The H-DRLN leverages a deep neural network to model the policy and value functions, resulting in high effectiveness across a variety of Minecraft tasks such as navigation, mining, and combat. Despite its success, this approach requires intensive training time and a less restrictive environment for it to be successful. Thus, it is less suitable for our context.

Problem Definition

Minecraft is an open-world game without an explicit goal. In this work, we take as a case study the Craft Wooden Pogo task, as defined in the PAL Minecraft environment. (Goss et al. 2023). In this task, the Minecraft agent, colloquially called Steve, is located in a field comprising $N \times N$ blocks and surrounded by unbreakable bedrock walls. The field includes multiple trees and a crafting table. Steve is tasked with crafting a pogo stick, which requires performing the following actions (illustrated in Figure 1):

1. Harvest at least three wood blocks from trees.

[3]https://github.com/PolycraftWorld/PAL

Figure 1: A plan to accomplish the Craft Wooden Pogo task.

2. Use the wood to craft planks.
3. Use planks to craft sticks.
4. Use some of the sticks and planks to craft a tree tap
5. Place a tree tap near a tree to collect polyisoprene sacks.
6. Use the remaining sticks, planks, and polyisoprene sacks to craft a wooden Pogo stick.

The original task in PAL was predefined with a fixed map size and predetermined positions for the trees, the crafting table, and the agent. Moreover, the agent always starts with an empty inventory. To introduce variability across problem instances, we developed a problem generator that generates initial states in which it randomly assigns: (1) the agent's starting position on the map, (2) the quantity and placement of trees, (3) the items present in the agent's inventory, and (4) the crafting table's position. Thus, different sequences of actions are needed to solve different problem instances.

We consider the problem of accomplishing the Craft Wooden Pogo task in two settings: *offline with expert demonstrations* and *online*.

Offline Learning from Expert Demonstrations. In this setting, our agent is given a set of *trajectories* created by observing an *expert* solve the Craft Wooden Pogo task, i.e., successfully craft the wooden pogo stick. The agent objective in this setting is to use the set of expert trajectories and output a *plan* or a *policy* specifying how to act in order to accomplish the Craft Wooden Pogo task. Both learning and planning in this setting are done *offline*, that is, after processing the expert trajectories and outputting a plan or a policy, the agent executes the actions in the plan or the policy until either the task is accomplished or not. Thus, this setting corresponds to the Offline RL setting (Sutton and Barto 2018) and IL (Bratko, Urbančič, and Sammut 1995). The main measure we consider in this setting is *success rate* for a given number of expert trajectories, i.e., the number of game episodes our agent can solve after it is given these expert trajectories.

Online Learning. In this setting, the agent must perform actions in the environment to explore it, and no expert trajectories are given. Specifically, the agent interacts with the environment in a sequence of *episodes*. Every episode starts from some initial state of the environment and ends either when the agent successfully crafts the wooden pogo stick or when the agent executes more actions than a predetermined number of actions. Note that, unlike the offline setting, here planning and learning must be interleaved. In each episode, the agent undertakes the task of planning which

actions to execute and subsequently engages in the learning process to enhance performance based on the outcomes of those actions. This configuration aligns with the conventional RL setting. The primary metric of interest in this context is the cumulative reward, quantified by tallying the number of episodes in which the agent successfully completes the task. Additionally, we are interested in gauging the time to convergence, denoted by the number of steps it takes for the agent to solve newly presented problem instances.

Solving the Craft Wooden Pogo Task

For each of the problem settings, i.e., offline and online, we propose two approaches: one based on RL techniques and the other on domain model learning and planning.

Offline Learning from Expert Observations

For the RL-based approach, this setting can be viewed as either an Offline RL (Kumar et al. 2020) or IL (Bratko, Urbančič, and Sammut 1995). Thus, either DQN, BC, or GAIL may be used as is.

For the planning-based approach, we propose providing the expert observations as input trajectories to N-SAM, which outputs a PDDL domain model of the environment. This PDDL domain is used as input to an off-the-shelf domain-independent numeric planner, that solves the resulting planning problem and outputs a plan for crafting the wooden pogo task. If the planner cannot solve the resulting problem, failure is declared. While any domain-independent numeric planner can be used to solve the resulting planning problem, we used Metric-FF (Hoffmann 2003), a state-of-the-art numeric planner. The agent derived from this approach is henceforth referred to as the *PDDL-agent*. We consider two types of PDDL-agents. The first does not know which variables are relevant to each action. The second is given this additional information, i.e., it knows which variables are involved in each action's preconditions and effects. We refer to this agent as PDDL-agent with Relevant Variables (PDDL-agent_RV). Note that PDDL-agent_RV does not know the actions' preconditions and effects; it only knows the variables they refer to.

Online Learning

This setting adheres to the classical RL setting: an agent engages in actions within the environment, receives observations, and adapts its behavior over time. Thus, we used standard RL techniques, namely DQN and PPO. To leverage the deterministic nature of the domain, we used PPO with action masking (Tang et al. 2020) instead of regular PPO. This version of PPO prevents executing actions that were previously attempted in the current state and deemed inapplicable.

Our planning-based approach tailored for the offline case lacks direct applicability to the online setting due to the absence of a mechanism for collecting trajectories, essential in this context. To address this, we introduce a novel hybrid strategy that integrates planning and RL, as follows. At the beginning of each episode, we deploy N-SAM to construct an action model based on all previously gathered trajectories.

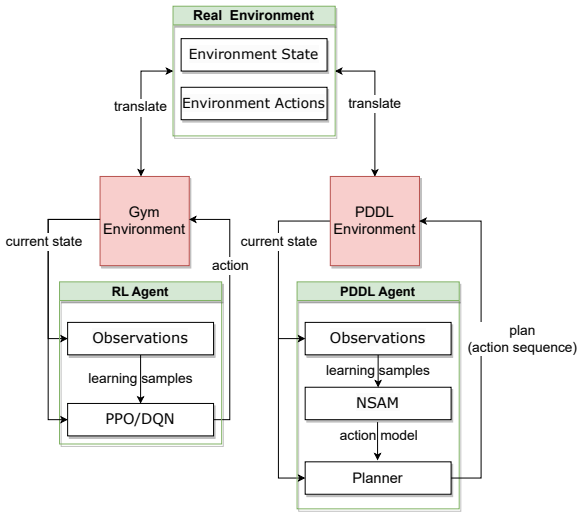


Figure 2: Modeling environment interaction.

Then, we use a planner (we used Metric-FF) with the generated action model to try to find a plan for the current episode. If successful, the agent executes the plan. Otherwise, an RL algorithm (we used PPO) is used to choose actions throughout the episode. This integration establishes a symbiotic relationship between RL and planning, fostering dynamic interaction and mutual benefits. RL acts as a methodological tool to solve problems and gather information when planning fails, leveraging its inherent ability to balance exploration and exploitation. This balance serves two purposes: attempting to directly solve the problem and collecting goal-oriented observations for action model learning, enhancing problem-solving efficiency.

Simultaneously, RL benefits from this partnership by using plan-executed trajectories as a training ground. These trajectories guarantee problem-solving success, providing higher-quality data that improves sample efficiency and stabilizes the learning process (Sutton and Barto 2018). This holistic approach allows for efficient adaptation of automated planning to online learning, potentially surpassing standard RL techniques.

It is worth noting that in the online setting, the agent may execute actions in states where they are not applicable. For instance, the agent might try to break a tree when it is not in proximity to a tree block. N-SAM is geared towards learning only from successfully executed actions and ignores such transitions. Future work can extend N-SAM to exploit this information as well.

Modeling the Craft Wooden Pogo Task

Efficient modeling and knowledge representation are key to solving hard learning and planning tasks. This work explores two alternative approaches to model the Craft Wooden Pogo task. Each model incorporates translation mechanisms that convert Minecraft states and actions into both PDDL 2.1, utilized by the planner, and an AI gym environment (Brockman et al. 2016), employed by the RL agents. The interac-

tion between the agent and the environment is depicted in Figure 2.

Item Counts Model

We first propose a modeling of the Craft Wooden Pogo that ignores the locations of items on the grid map, i.e., the agent observes the number of trees available as well as the quantities of the other ingredients in the inventory. We refer to this modeling approach as *Item Counts*. Item Counts is a simplification of the original task since the agent is not required to explore the map and locate the trees. This allows us to define higher level *macro actions* for the agent to choose from and correspondingly define a more compact state representation. Macro actions allow agents to optimize their gameplay and reduce the amount of time spent on repetitive tasks. It allows the agent to make more strategic decisions and react quickly to changing circumstances. OpenAI's use of macro actions in Starcraft (2019) is an excellent example of how this approach can lead to more competitive and engaging gameplay.

In our context, we define the following macro actions that encapsulate multiple lower-level PAL actions:

1. **GET_LOG** - executes teleportation to a tree, breaking it, collecting the logs, and adding them to the inventory.
2. **CRAFT_PLANK** - craft planks from the logs in the inventory.
3. **CRAFT_STICK** - craft sticks from the planks in the inventory.
4. **CRAFT_TREE_TAP** - teleport to the crafting table, craft one tree tap, and add it to the inventory.
5. **PLACE_TREE_TAP** - teleport to a tree, move left, place the tree tap on it, collect the sack of polyisoprene, and add it to the inventory.
6. **CRAFT_WOODEN_POGO** - teleport to the crafting table, craft a wooden Pogo stick and add it to the inventory.

In this model, none of the actions require any parameters. Thus, the overall branching factor is 6. A state in this model includes the number of trees in the map, as well as the number of items of each type in the agent's inventory, a total of 7 state variables. Within the gym environment framework, the agent is rewarded with a score of 1 upon successful completion of the task, and it receives a reward of 0 otherwise.

All Blocks Model

In the second modeling approach, the agent receives the map data as input, as well as the number of items it is currently holding in the inventory.

The actions the agent can perform in the All Blocks modeling approach:

1. **TP_TO** - teleport from the current location to another cell on the map.
2. **BREAK**- breaks a tree to extract and add the logs to the inventory.
3. **CRAFT_PLANK** - craft planks from the logs in the inventory.
4. **CRAFT_STICK** - craft sticks from the planks in the inventory.
5. **CRAFT_TREE_TAP** - teleport to the crafting table, craft one tree tap, and add it to the inventory.
6. **PLACE_TREE_TAP** - when in front of a tree, move left, place the tree tap on it, collect the polyisoprene sack, and add it to the inventory.
7. **CRAFT_WOODEN_POGO** - teleport to the crafting table, craft a wooden pogo stick, and add it to the inventory.

These settings present a difficulty for the agents since the action spaces can be very large. In PDDL, the TP action involves two parameters: the current position of the agent and its target position. Similarly, actions such as break, place tree tap, craft tree tap, and craft pogo, also require the current position of the agent. Consequently, the total number of grounded actions in PDDL is $N^4 + 4N^2 + 2$, where N represents the width and height of the map. On the other hand, in the gym environment, the current location of the agent is not a requisite parameter, resulting in a total number of actions equal to $N^2 + 6$. The state in each model, illustrated in Figure 3, encompasses the entire map, detailing the cell type at each location, along with the agent's inventory and position. Additionally, in the gym environment, there's an explicit specification of the cell type directly in front of the agent. This inclusion is designed to assist the RL algorithms. Finally, similar to the Item Counts model, the agent receives a reward of 1 if and only if it successfully completes the task.

```
(define (domain PolyCraftFull)
    (:types
        cell - object
    )
    (:constants
        crafting_table - cell
    )
    (:predicates
        (position ?c - cell)
        ; Map
        (tree_cell ?c - cell)
        (air_cell ?c - cell)
        (crafting_table_cell ?c - cell)
        (has_pogo_stick)
    )
    (:functions
        ; Items in inventory
        (count_log)
        (count_planks)
        (count_sticks)
        (count_sacks)
        (count_tree_taps)
    )
```

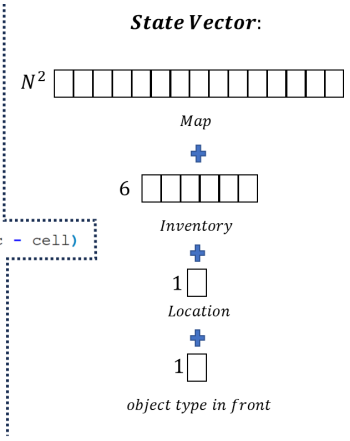
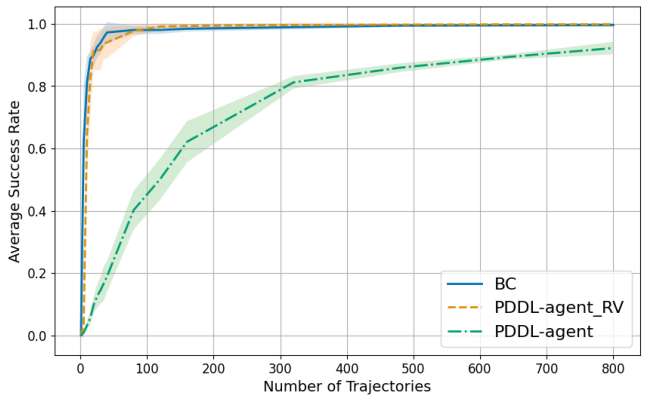


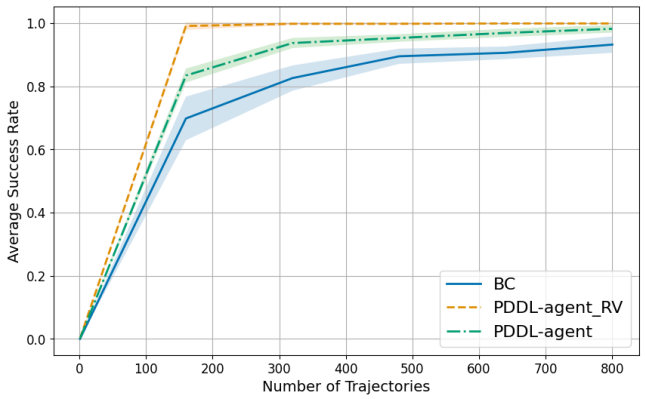
Figure 3: State representation of the All Blocks model: PDDL (left), RL (right).

Experimental Results

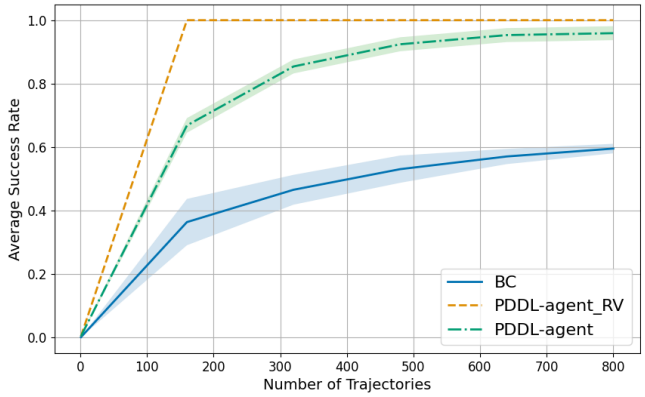
In this section, we describe an experimental evaluation of our approach. For each experiment, we created 1000 tasks. We configured the number of items in the inventory to range from zero to eight for all items except for the polyisoprene sack and the Pogo stick, which were always zero. We set the number of trees on each map to range from zero to (map size)/3.



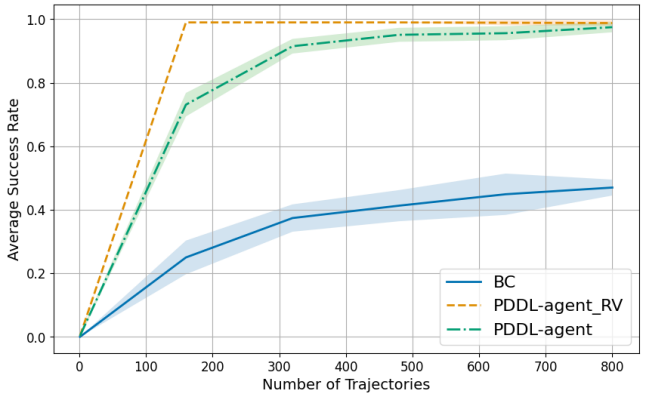
(a) Item Counts model



(b) All Blocks model with 6×6 maps



(c) All Blocks model with 10×10 maps



(d) All Blocks model with 15×15 maps

Figure 4: Results of offline learning from expert observations.

Offline Experiments

We conducted experiments to compare our PDDL-agents with DQN, representing offline RL algorithms, and the imitation learning algorithms BC and GAIL. The algorithms were trained on trajectories generated by an expert agent capable of solving the tasks. This expert agent was constructed by manually modeling the task as a planning problem and subsequently employing a planner to find solutions. Each plan generated was validated within the environment.

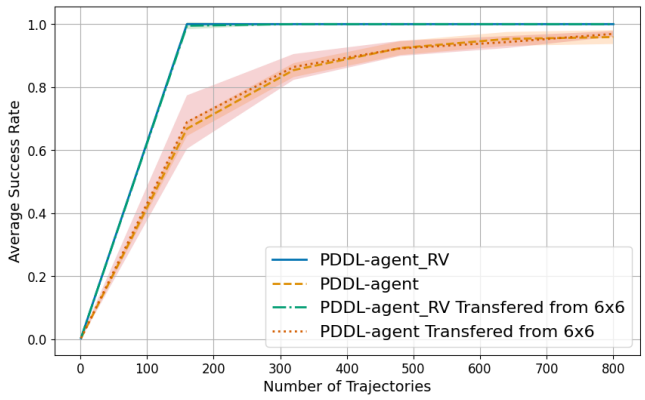
To assess the algorithms, we divided our 1000 tasks into training and test sets with an 8:2 ratio, respectively. The agents were then trained using the examples from the training set, and the learning process was evaluated by deploying the trained agent on the test set. To ensure robustness and generalization, we repeated this process in a 5-fold cross-validation framework, calculating the average and standard deviation over the five folds.

RL Training Configuration. We implemented our RL algorithms using open-source code. Specifically, we used the implementation of BC and GAIL available in the *imitation* library[4] and the implementation of DQN available in the *stable_baselines3* library [5]. For the neural network architecture, we configured all models as fully connected neural networks with sizes $512 \times 256 \times 256$, consisting of three layers. The first layer has 512 units, while the second and third layers have 256 units each. The hyperbolic tangent activation function was chosen for each layer.

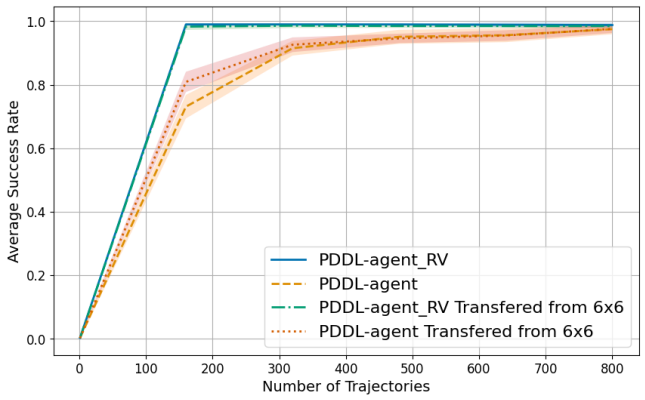
Results. In Figure 4, the average success rates are depicted as a function of the number of trajectories. Each subfigure corresponds to a different model and/or map size. The average across the 5 folds is represented by a line, while the corresponding standard deviation is visualized as a shaded area with low opacity around each line. Despite exhaustive efforts in hyperparameter tuning and experimentation with various network architectures, both DQN and GAIL consistently failed to solve the Pogo crafting task across all experiments. As a result, their performance is omitted from the presented plots. In the Item Counts model (Figure 4a), we observe that BC consistently solves the problem after a minimal number of trajectories, demonstrating performance comparable to that of PDDL-agent_RV. On the contrary, the PDDL-agent without the RV assumption requires more trajectories to effectively learn the action model. However, in

[4]https://imitation.readthedocs.io

[5]https://stable-baselines3.readthedocs.io



(a) Zero-shot transfer: All Blocks from a 6×6 map to 10×10.



(b) Zero-shot transfer: All Blocks from a 6×6 map to 15×15.

Figure 5: Results of offline transfer learning from expert observations.

the All Blocks model, the efficacy of BC diminishes, achieving lower success rates as the map size increases (refer to Figures 4b, 4c, and 4d). In contrast, the PDDL-agent, both with and without the RV assumption, consistently solves the problem with only a few hundred given trajectories.

Zero-shot Transfer. A key advantage of planning lies in its capacity to generalize across varying numbers of objects. In contrast, reinforcement learning algorithms do not inherently possess the same capability.[6]

To assess the generalizability of the PDDL-agents, we conducted a zero-shot transfer evaluation. The agents were trained on the All Blocks model with a 6×6 map and subsequently evaluated on larger map sizes (10×10 in Figure 5a and 15×15 in Figure 5b). The results demonstrate that both PDDL-agents achieved performance comparable to agents directly trained on the corresponding maps, effectively demonstrating perfect zero-shot generalization.

Online Experiments

In the online experiments, we conducted a comparative analysis between our hybrid agent—formed by combining the PDDL-agent and PPO —and the standalone PPO with action masking. The training phase encompassed 25 tasks. For the Item Counts model, each task had a budget of 400 steps, with a maximum episode length of 100. However, the All Blocks model with a 6×6 map, featuring a larger action and state space, required additional exploration. Consequently, each task in the All Blocks model was allocated a budget of 2000 steps, with a maximum episode length of 500. This resulted in each task being repeated at least four times both the Item Counts model and the All Blocks model, ensuring thorough learning and exploration.

[6]Although architectures such as Graph Neural Networks offer a limited form of generalization, achieving this requires non-trivial engineering efforts and often results in degraded performance when the number of objects changes (Munikoti et al. 2023).

Reinforcement Learning Configurations

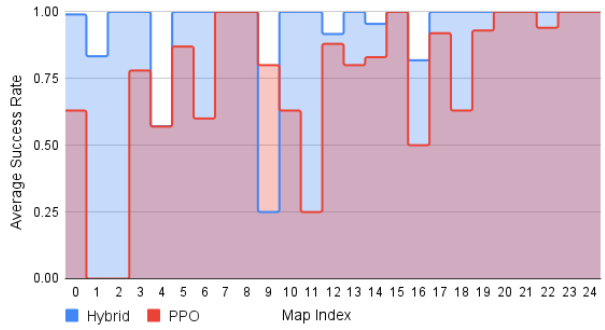
We used the standard models from the python library `stable_baselines3`[5]. The PPO network architecture is a fully connected neural network of size 64×64, i.e., two layers with 64 units, and the selected activation function for each layer is $tanh$. This network was trained with the following configuration: an entropy coefficient of 0.01, a discount factor of 0.999, a value function coefficient of 0.65, and a maximum gradient clipping of 1.0.

Results. Figure 6a illustrates the mean episodic reward achieved in each map on the Item Counts model, while Figure 6c presents the results for the All Blocks model with a 6×6 map. The results suggest that the hybrid approach is capable of solving the task more consistently compared to PPO, showcasing an enhanced and progressively improving performance over time. Figures 6b and 6d display the mean episode length for both algorithms on the two models. It is evident that the hybrid approach leads to significantly shorter episodes when the planner successfully finds a plan. In fact, the typical plan discovered is an order of magnitude shorter than a solution generated by PPO.

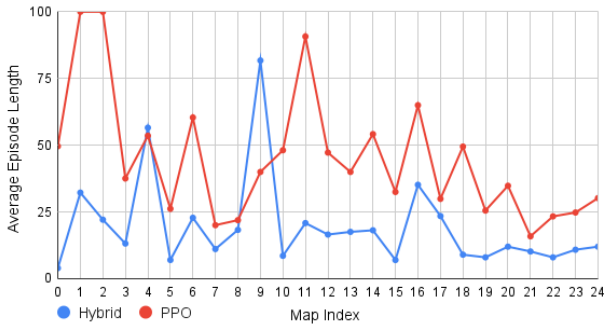
Conclusions and Future Work

In this work, we proposed a planning-based agent called PDDL-agent for solving Minecraft tasks in two settings: offline learning with expert observations and online learning. PDDL-agent uses N-SAM (Mordoch, Juba, and Stern 2023) to learn an action model of the environment, and uses a state-of-the-art planning algorithm to solve the Minecraft task at hand. For the online setting, PDDL-agent employs a novel hybrid action-selection strategy that uses both planning and RL. As a case study, we considered the Craft Wooden Pogo task and proposed two ways to represent states and actions in this domain, namely Item Counts and All Blocks. We compared our PDDL-agent to several RL-based agents experimentally when using Item Counts and when using All Blocks, under the offline setting and the online setting.

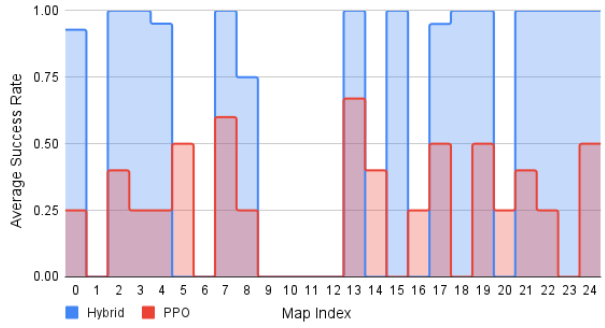
Experimental evaluation showed the benefit of our PDDL-agent outperforming all RL-based approaches in most cases.



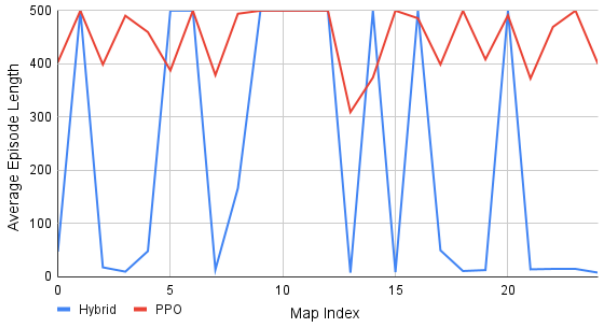
(a) Avg. success rate, Item Counts model, higher is better



(b) Avg. episode length, Item Counts, lower is better



(c) Avg. success rate, All Blocks, 6×6 maps, higher is better



(d) Avg. episode length, All Blocks, 6×6 maps, lower is better

Figure 6: Results of online learning.

These results highlight the potential benefit of using action model learning and planning over purely RL-based methods. Also, it shows the benefit of the proposed hybrid strategy for interleaving action model learning with goal-oriented RL. In future work, we aim to enhance the action selection process for online learning. Our approach involves exposing PPO to diverse plans for solving each task, offering a variety of high-quality samples, rather than relying on a single plan. Additionally, we plan to pioneer a novel online approach that prioritizes selecting actions aimed at improving the learned action model. This marks a departure from the goal-oriented approach employed in this research.

References

Aeronautiques, C.; Howe, A.; Knoblock, C.; McDermott, I. D.; Ram, A.; Veloso, M.; Weld, D.; SRI, D. W.; Barrett, A.; Christianson, D.; et al. 1998. PDDL— The Planning Domain Definition Language. *Technical Report, Tech. Rep.*

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Amir, E.; and Chang, A. 2008. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33: 349–402.

Bartheye, O.; and Jacopin, E. 2009. A real-time PDDL-based planning component for video games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 5, 130–135.

Bratko, I.; Urbančič, T.; and Sammut, C. 1995. Behavioural cloning: phenomena, results and problems. *IFAC Proceedings Volumes*, 28(21): 143–149.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Cresswell, S.; McCluskey, T.; and West, M. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, 28(2): 195–213.

Duarte, F. F.; Lau, N.; Pereira, A.; and Reis, L. P. 2020. A survey of planning and learning in games. *Applied Sciences*, 10(13): 4529.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20: 61–124.

Frazier, S.; and Riedl, M. 2019. Improving deep reinforcement learning in minecraft with action advice. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 15, 146–152.

Georgievski, I.; and Aiello, M. 2015. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222: 124–156.

Goss, S. A.; Steininger, R. J.; Narayanan, D.; Olivença, D. V.; Sun, Y.; Qiu, P.; Amato, J.; Voit, E. O.; Voit, W. E.; and Kildebeck, E. J. 2023. Polycraft World AI Lab (PAL): An Extensible Platform for Evaluating Artificial Intelligence Agents. *arXiv preprint arXiv:2301.11891*.

Guss, W. H.; Codel, C.; Hofmann, K.; Houghton, B.; Kuno, N.; Milani, S.; Mohanty, S.; Liebana, D. P.; Salakhutdinov, R.; Topin, N.; et al. 2019a. NeurIPS 2019 competition: the MineRL competition on sample efficient reinforcement learning using human priors. *arXiv preprint arXiv:1904.10079*.

Guss, W. H.; Houghton, B.; Topin, N.; Wang, P.; Codel, C.; Veloso, M.; and Salakhutdinov, R. 2019b. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*.

Ho, J.; and Ermon, S. 2016. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research*, 20: 291–341.

Hoffmann, J.; and Magazzeni, D. 2019. Explainable AI planning (XAIP): overview and the case of contrastive explanation. *Reasoning Web. Explainable Artificial Intelligence*, 277–282.

Hussein, A.; Gaber, M. M.; Elyan, E.; and Jayne, C. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2): 1–35.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 379–389.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Mordoch, A.; Juba, B.; and Stern, R. 2023. Learning Safe Numeric Action Models. In *AAAI*, 12079–12086. AAAI Press.

Mordoch, A.; Shperberg, S. S.; Stern, R.; and Juba, B. 2023. Enhancing Numeric-SAM for Learning with Few Observations. *CoRR*, abs/2312.08700.

Munikoti, S.; Agarwal, D.; Das, L.; Halappanavar, M.; and Natarajan, B. 2023. Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications. *IEEE Transactions on Neural Networks and Learning Systems*.

Palucka, T. 2017. Polycraft World teaches science through an endlessly expansive universe of virtual gaming: https://polycraft.utdallas.edu. *MRS Bulletin*, 42(1): 15–17.

Pomerleau, D. A. 1991. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1): 88–97.

Roberts, M.; Piotrowski, W.; Bevan, P.; Aha, D.; Fox, M.; Long, D.; and Magazzeni, D. 2017. Automated planning with goal reasoning in Minecraft. In *ICAPS workshop on Integrated Execution of Planning and Acting (IntEx)*.

Scheller, C.; Schraner, Y.; and Vogel, M. 2020. Sample efficient reinforcement learning through learning from demonstrations in minecraft. In *NeurIPS 2019 Competition and Demonstration Track*, 67–76. PMLR.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Segura-Muros, J. Á.; Pérez, R.; and Fernández-Olivares, J. 2021. Discovering relational and numerical expressions from plan traces for learning action models. *Applied Intelligence*, 1–17.

Stern, R.; and Juba, B. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Tang, C.; Liu, C.; Chen, W.; and You, S. D. 2020. Implementing action mask in proximal policy optimization (PPO) algorithm. *ICT Express*, 6(3): 200–203.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D.; and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354.

Wang, X. 1994. Learning planning operators by observation and practice. In *Second International Conference on Artificial Intelligence Planning Systems (AIPS)*, 335–340.

Watkins, C. J. C. H. 1989. *Learning from delayed rewards*. Ph.D. thesis, Oxford: King's College.

Wichlacz, J.; Torralba, A.; and Hoffmann, J. 2019. Construction-planning models in minecraft. In *Proceedings of the ICAPS Workshop on Hierarchical Planning*, 1–5.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3): 107–143.