RAMP: Hybrid DRL for Online Learning of Numeric Action Models

Yarin Benyamin
Ben-Gurion University of the Negev
Be'er Sheva, Israel
bnyamin@post.bgu.ac.il

Argaman Mordoch
Ben-Gurion University of the Negev
Be'er Sheva, Israel
mordocha@post.bgu.ac.il

Shahaf Shperberg
Ben-Gurion University of the Negev
Be'er Sheva, Israel
shperbsh@bgu.ac.il

Roni Stern
Ben-Gurion University of the Negev
Be'er Sheva, Israel
roni.stern@gmail.com

ABSTRACT

Automated planning algorithms require an *action model* specifying the preconditions and effects of each action, but obtaining such a model is often hard. Learning action models from observations is feasible, but existing algorithms for numeric domains are offline, requiring expert traces as input. We propose the Reinforcement learning, Action Model learning, and Planning (RAMP) strategy for learning numeric planning action models online via interactions with the environment. RAMP simultaneously trains a Deep Reinforcement Learning (DRL) policy, learns a numeric action model from past interactions, and uses that model to plan future actions when possible. These components form a positive feedback loop: the RL policy gathers data to refine the action model, while the planner generates plans to continue training the RL policy. To facilitate this integration of RL and numeric planning, we developed Numeric PDDLGym, an automated framework for converting numeric planning problems to Gym environments. Experimental results on standard IPC numeric domains show that RAMP significantly outperforms PPO, a well-known DRL algorithm, in terms of solvability and plan quality.

KEYWORDS

Deep Reinforcement Learning; Numeric Planning; Action Model Learning; Online Learning;

1 INTRODUCTION

Automated planning [9] is a reliable approach for sequential decision-making, but it requires an *action model* specifying the preconditions and effects of each action. Handcrafting action models is notoriously difficult, especially for *numeric planning problems*, where action preconditions and effects involve discrete and continuous state variables. Prior work proposed automated methods for action model learning (AML) in numeric planning [27, 35] were *offline*, requiring experts to provide execution traces to learn from. We consider the *online learning* setting, where an agent must learn from its own interactions with the environment. Deep Reinforcement Learning (DRL) algorithms such as Proximal Policy Optimization (PPO) [34] are designed for such online learning settings, but they often lack the structural advantages of symbolic planning.

Proc. of the Adaptive and Learning Agents Workshop (ALA 2026), Aydeniz, Delgrange, Mohammedalamen, Yang (eds.), May 25 – 26, 2026, Paphos, Cyprus, https://alaworkshop2026.github.io/. 2026.

In this work, we present RAMP, a novel strategy for solving numeric planning problems that integrates DRL, online AML, and planning. RAMP executes a positive feedback loop: an RL policy is used to explore the environment and collect data in a goal-oriented manner. This data is used to learn a numeric planning action model. This learned model is then utilized by a planner to generate high-quality plans, which in turn accelerate the training of the DRL policy. To enable this DRL-planning integration, we developed a framework for automatically converting numeric planning domains specified in PDDL 2.1 [8] into standard Gym environments [42].

We implemented RAMP and evaluated empirically on three domains from the International Planning Competition (IPC) and a recently proposed numeric planning domain inspired by Minecraft [3]. Results show that RAMP is able to learn effective action models and use them to solve more problems and find better solutions than PPO [34], a state-of-the-art DRL baseline.

2 BACKGROUND

A *numeric planning domain* in PDDL2.1 [8] is defined by a tuple $D = (A, T, F, X, P)$, where A is a set of actions, T is a set of object types, F is a set of fluents, and X is the set of functions. A fluent is a Boolean variable, while a function is a numeric variable. Every action in A is associated with a set of *preconditions* (logical and linear numeric constraints) and *effects* (assignments or modifications to the fluents and functions). Domains are usually defined in a *lifted* manner, which means that the fluents, functions, and actions are parameterized by object types. A *numeric planning problem* in a domain D is defined by a tuple $P = (s_0, O, G)$, where s_0 is an initial state, O is a set of objects, and G is a goal definition. A grounded fluent is a fluent with specific objects in O assigned to its parameters. Grounded functions and actions are similarly defined. A state in a numeric planning problem is an assignment of values to all the grounded fluents and functions. A goal in numeric planning is a set of constraints over the grounded fluents and functions, and a goal state is any state that satisfies the goal constraints. A solution to a planning problem is a *plan*, which is an applicable sequence of actions leading from an initial state to a goal state.

Learning Domain Models . Automated planning algorithms assume the existence of a symbolic model. However, obtaining such a model is challenging, motivating the development of automated Action Model Learning (AML) algorithms. Research has primarily focused on the *offline* setting, where the AML algorithm is given

a set of *trajectories* to learn from. A trajectory is an alternating sequence of states and actions, where the state after an action is the result of applying the action on the state immediately before it.

In the *online* learning setting, no expert trajectories are given. Instead, an agent interacts with the environment in a sequence of episodes to maximize cumulative reward or minimize plan length. While several online learning algorithms have been proposed for classical planning domains, there are currently no methods for online learning of numeric planning domains.

A critical property of some AML algorithms is that they provide a form of "safety" guarantee: not only must the learned model be consistent with the given observations, but every plan generated from it is guaranteed to be *sound* with respect to the true, unknown action model.

Definition 1 (Safe Domain Model). *Let M^* be the true, unknown action model of an environment, and let M be a learned domain model. M is considered* safe *if every plan that is valid in M is* sound *with respect to M^*. Therefore, for any initial state s_0 and goal G, if a plan $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ reaches G under the transition dynamics of M, it is fully executable under M^* from s_0, and its execution in M^* results in a state s_n that satisfies G.*

Reinforcement Learning (RL) . Reinforcement Learning (RL) [40] is a framework for solving decision-making problems where an agent learns a policy $\pi(a|s)$—mapping a state s to an action a—to maximize expected cumulative rewards. Deep RL (DRL) extends this by training neural networks to represent the policy. A notable RL algorithm, which we make use of in this work, is Proximal Policy Optimization (PPO) [34], a stable policy-gradient algorithm that provides state-of-the-art performance in discrete decision-making environments. It operates by alternating between sampling data through environmental interaction and optimizing a surrogate objective function, utilizing a clipping mechanism to prevent destructively large policy updates and ensure stable learning.

Standard DRL algorithms are known to struggle in planning problems that require reasoning over long horizons. Also, most DRL algorithms are designed for richer environments than numeric planning, including stochastic effects and raw observations such as images. Thus, they are expected to perform poorly on problems that can be adequately abstracted as symbolic numeric planning problems. Hybrid RL-symbolic approaches like SK [37] and SORL [17] have been proposed, but do not address symbolic numeric planning.

Our work addresses this gap by introducing RAMP, the first strategy for *online* learning of *numeric* action models, integrating safe model learning with Deep RL.

Problem Setting. The agent is situated in an environment modeled as a numeric planning domain D. The agent has access to the domain's types, fluents, functions, and action names, but not the domain's action model, i.e., it does not know the actions' preconditions and effects. In every *episode*, a problem $p = (s, O, G)$ in this domain is given to solve. The agent then chooses which action to perform, leading to a new state s'. This process is repeated until the problem is solved, i.e., s' satisfies the goal G, or until reaching a maximum number of steps t_{MAX}. The objective is to learn a policy π that solves problems efficiently, i.e., that guides the agent to a goal with as few steps as possible.

3 RELATED WORK

In this section, we discuss several lines of research that are related to this work.

3.1 Offline Action Model Learning Algorithms

FAMA [1] is an offline action model learning algorithm that can handle missing observations and outputs a classical planning domain model. It frames the task of learning an action model as a planning problem, ensuring that the returned action model is consistent with the provided observations. NOLAM [22] can learn action models even from noisy trajectories. LOCM [7] learns an action model from observed sequences of actions and their signatures, without observing the states in the trajectory.

The Safe Action Model (SAM) learning algorithm [18, 38] differs from the above algorithm in that the action model it returns provides a form of "safety" guarantee (Definition 1). SAM has been extended to support lifted action model representation [18], partial observability [23], stochastic effects [19], and conditional effects [28]. All SAM algorithms require observing all the actions in the given trajectories, and most also require observing the states. *NSAM* [27], the algorithm we use in this work, extends SAM to numeric domains while preserving the same safety guarantees.

While the aforementioned algorithms rely on structured execution traces, other approaches extract models from less structured knowledge. Framer [25] induces planning models by clustering semantic frames in textual activity descriptions. In contrast, Large Language Models (LLMs) leverage pre-trained knowledge to construct models from language and can support interactive planning workflows for execution and validation [4, 41]. While these approaches reduce the barrier to model creation, they do not address online learning of numeric action dynamics.

To handle environments where neither symbolic nor language inputs are available, recent work has explored learning action models directly from raw images. LatPlan [2] learns propositional action models in the latent space using a variational autoencoder. They use the Gumbel-Softmax technique [16] to convert the continuous output of an autoencoder into categorical variables. These categorical variables are used as propositional symbols in a symbolic reasoning system, which, in LatPLan's case, is a symbolic action model. ROSAME-I [47], like LatPlan, learns action models from visual inputs. Unlike LatPlan, ROSAME-I requires knowing the set of possible propositions and action signatures as input. ROSAME-I simultaneously learns classifiers for identifying propositions in a given image and infers a lifted, first-order action model defined over the given set of propositions and actions.

Although these approaches handle symbolic or unstructured inputs, they do not specifically address numeric planning problems. NSAM [27] and PlanMiner [35] are, to the best of our knowledge, the only algorithms capable of learning action models that include both discrete and numeric preconditions and effects.[1] NSAM makes several simplifying assumptions, full, noise-free observability, conjunctions of linear inequalities for preconditions, and conjunctions of linear equations for effects, which allow it to run in polynomial time. Under these assumptions, it guarantees a *safe domain model*

[1] Some other action model learning algorithms are capable of learning the numeric costs or rewards associated with actions [17].

by computing the minimal numeric preconditions via convex hulls over successful states and exact numeric effects through linear equation solving. In contrast, PlanMiner can handle noisy observations but does not provide safety guarantees and requires solving a generally intractable symbolic regression problem. For these reasons, we selected NSAM as our primary action model learning algorithm.

3.2 Online Action Model Learning Algorithms

Online action-model learning algorithms iteratively learn an incumbent action model and choose the next actions to perform in order to collect observations that enable further refinement of the incumbent action model. OLAM [21] is an online action model learning algorithm that is designed for classical planning domains. It identifies in every iteration an action and a state where trying to execute that action is expected to refine the incumbent action model. Then, it uses a planner to find a plan to reach that state and attempts to execute the chosen action. GLIB [6] follows a similar approach but is designed for stochastic environments, resulting in a Probabilistic PDDL (PPDDL) [49] action model. QACE [43] is an action model learning algorithm that can also query a black-box expert. It outputs a PPDDL action model with the same capabilities as the black-box expert it trained from. Karia et al. 2023 extend QACE to address the non-stationarity of the environment, i.e., address cases where the environment dynamics change. QACE+ achieves this by interleaving planning and learning and focusing on learning only the models essential for the tasks at hand. ILM [30] employs an explore-exploit strategy: if it reaches a state from which the goal can be achieved, it exploits this state; otherwise, it explores through random walks. Instead of focusing solely on reaching a specific goal, the agent can take a broader approach by exploring the environment and aiming for an interesting state in it.

Recent works explored integrating Reinforcement Learning (RL) and online learning of a symbolic action model [17, 37]. The objective of these works is typically to maximize a cumulative reward metric, in contrast to action model learning algorithms like OLAM and ILM, whose objective is to learn a symbolic action model. Sreedharan and Katz [37] proposed such an algorithm, which we refer to as the SK algorithm. SK begins by initializing an optimistic symbolic model that assumes all actions are applicable in every state (i.e., no preconditions) and the effect of each action includes all grounded predicates. It then employs fast, diverse planners to generate potential paths toward the goal. While these paths are unlikely to be valid, they serve as exploration mechanisms to gather new information. Specifically, these plans are executed within the environment, with the outcomes used to train a reward-maximizing policy using Q-learning [45, 46]. This continuous process of exploration and symbolic model refinement is guaranteed to generate a goal-reaching policy. Our hybrid strategy for the online learning setting is somewhat similar to SK. However, SK is only designed for classical (non-numeric) planning, and its applicability to numeric planning remains uncertain. SORL [17] is another online algorithm that integrates RL and learning symbolic action models to maximize the cumulative reward. It collects visual observations from the environment and assumes the existence of a mapping function from visual observations to symbolic states. SORL iteratively learns and creates a symbolic, higher-level action model, and a lower-level set of RL policies, referred to as *symbolic options*. A planner uses the learned symbolic action model to create a high-level plan, and a meta-controller chooses or creates symbolic options to try to execute the high-level plan, exploring the environment as needed.

While not explicitly specified, the action model learning algorithm SORL uses is not robust to missing or noisy observations. It assumes that an action's effects can be inferred by the difference between the states observed before and after applying that action and shows no support for numeric preconditions. Numeric effects are supported in a very limited way, only learning which actions increase the reward and by how much. Other numeric aspects, e.g., numeric state variables and preconditions, are not learned.

3.3 Summary: Action Model Learning Algorithms

Table 1: Comparison of various action model learning algorithms, based on their support of given problem - numeric inputs, stochasticity, non-stationarity, noisy, observability, and online/offline learning capability.

	Input	Numeric	Stochastic	NS	Noisy	Obs.	Online/Offline
FAMA [1]	symbolic	No	No	No	No	Partial	Offline
LOCM [7]	symbolic	No	No	No	No	Only action	Offline
Framer [25]	text	No	No	No	-	Only action	Offline
OLAM [21]	symbolic	No	No	No	No	Yes	Online
NOLAM [22]	symbolic	No	No	No	Yes	Yes	Offline
ILM [30]	symbolic	No	Yes	Yes	Yes	Yes	Online
GLIB [6]	symbolic	No	Yes	No	No	Yes	Online
QACE [43]	symbolic	No	Yes	No	No	Yes	Online
QACE+ [20]	symbolic	No	Yes	Yes	No	Yes	Online
SORL [17]	visual[2]	No[3]	No	No	-	Yes	Online
SAM [18]	symbolic	No	No	No	No	Yes	Offline
NSAM [27]	symbolic	Yes	No	No	No	Yes	Offline
PlanMiner [35]	symbolic	Yes	No	No	Yes	Partial	Offline
SK [37]	symbolic	No	No	No	No	Yes	Online
JRK [15]	visual	No	Yes[4]	No	-	Yes	Offline
LATPLAN [2]	visual	No	No	No	-	Yes	Offline
ROSAME-I [47]	visual	No	No	No	-	No[5]	Offline
RAMP (our method)	symbolic	Yes	No	No	No	Yes	Online

Table 1 provides an overview of all action model learning algorithms described above. Every row represents a model-learning algorithm, and every column represents a property of action model-learning algorithms. Column "Input" refers to the type of input given to the learning algorithm, namely, whether it is symbolic, text or visual. Columns "Numeric" and "Stochastic" refer to whether the underlying environment includes numeric state variables and stochastic effects, respectively. Column "NS" (non-stationarity) refers to whether the dynamics of the underlying environment, i.e., the actions' preconditions and effects, may change during learning. Columns "Noisy" and "Obs." refer to whether the states and actions in the given observations are noisy and fully observed, respectively. Note that if the "Input" is visual, the algorithm can handle noise due to its use of function approximation for image processing. This is indicated by "-" in Table 1. The "Online/Offline" column refers to whether the learning algorithm is an online algorithm or an offline algorithm. As can be seen, the RAMP strategy we propose in this

[2]They assumed as input a perfect mapping from visual input to symbolic state.

[3]The support for numeric planning is limited to only learning how much reward each action adds.

[4]While they learn a PPDDL model, the experimental results all use a deterministic planner.

[5]Note that the first and last states in every trajectory are assumed to be fully obseravable

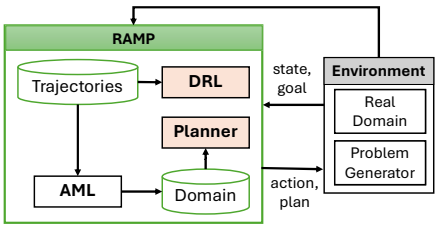

Figure 1: A high-level diagram of the RAMP strategy.

work is the only online learning algorithm that supports learning a numeric action model.

4 THE RAMP STRATEGY

RAMP integrates three components: a DRL algorithm, an AML algorithm, and a numeric planner. It maintains a set $\mathcal{T}$ of observed trajectories and an incumbent learned domain model, denoted M. Both are initialized to be empty. At the beginning of every episode, RAMP attempts to find a plan with the planner using M.[6] If a plan is found, the agent executes this plan. Otherwise, RAMP uses the DRL algorithm to choose which actions to perform in this episode. At the end of every episode, we add the resulting trajectory to $\mathcal{T}$. This trajectory is used as a training example for both the DRL algorithm and the AML algorithm. Re-running the offline AML on the cumulative trajectory set at the end of each episode is what effectively turns it into an online method. Figure 1 provides a high-level illustration of RAMP.

The integration of AML, planning, and DRL in RAMP establishes a symbiotic relationship between them. For the planner, the DRL algorithm acts as a fail-safe mechanism when it fails to find a plan. For the AML algorithm, the DRL algorithm provides a principled method for gathering observations in a goal-oriented manner, leveraging its inherent ability to balance exploration and exploitation. Simultaneously, the DRL algorithm uses the trajectories created by the planner for training. These trajectories tend to represent efficient ways to solve the problem. Such high-quality data improves sample efficiency and stabilizes the learning process, as we observed empirically in our experimental results.

Implementation Details . While RAMP is agnostic to the AML and DRL algorithms used, in our implementation, we use NSAM [7] for AML and the RLlib implementation of PPO for DRL. NSAM is one of only two existing AML algorithms that can learn numeric domains, and was chosen for its ability to return safe action models (Definition 1). PPO was used in our implementation of RAMP since it is a well-known, popular, stable, and has low sensitivity to hyperparameters. We have also considered alternative DRL algorithms such as Rainbow DQN [12] and Soft Actor-Critic (SAC) [11], but they consistently failed to learn across these continuous environments despite extensive hyperparameter tuning.

A challenge when using PPO is its *clipping mechanism*, which is designed to constrain policy updates by preventing the probability ratio between the new and old policies from deviating too far from 1. When the agent follows the plan generated by the planner, its own learned policy may assign a low probability to the dictated actions, creating a significant discrepancy between the executed actions and those preferred by the current policy. This can cause the importance-weighted policy ratio to frequently fall outside the clipping range, effectively nullifying the gradient update and slowing down learning. As a result, in some cases, PPO may struggle to meaningfully adjust its policy, particularly if the expert actions are substantially different from what it would naturally choose. To address this issue, we adopt an approach for masking invalid actions [14]. Specifically, we treat the expert action in each state as the only valid action and mask out all others. This ensures that the logits of actions that do not conform to the plan are set to zero, preventing them from influencing the policy update. As a result, the gradient for these actions is not eliminated, ensuring that the update is not affected by PPO's clipping mechanism.

Similarly, off-the-shelf DRL algorithms such as PPO are designed for stochastic environments. Thus, the agent may repeatedly attempt the same inapplicable action in a given state. To prevent this, we apply a *masking* mechanism that disallows actions previously observed to be inapplicable. Similar masking techniques have been used in DRL in other domains [5, 44, 48].

5 AUTOMATED PDDL TO GYM CONVERSION

RAMP requires the use of DRL algorithms to solve numeric PDDL planning problems. DRL algorithms work by interacting with the *environment*, which in our case is given by the PDDL problem. PDDLGym [36] provides a wrapper over a selected number of PDDL problems that allows simulating "interactions" with them and using RL algorithms to attempt to solve them. Specifically, they provide AI Gym [42] environments that simulate classical PDDL domains. This is particularly useful since standard implementations of RL algorithms, such as RLlib [24] and Stable Baselines [31], support the Gym interface. However, PDDLGym's design is primarily tailored to tabular RL methods and does not support numeric planning.

Therefore, we developed an automated framework to convert PDDL2.1 domains into AI Gym environments, which we refer to as Numeric PDDLGym.[8] This framework provides full support for Boolean and numeric state variables and lifted domain representation. It accepts a PDDL domain and problem file as input and generates a Gym environment that simulates the corresponding planning problem. A challenge in this conversion mechanism is that standard RL and DRL algorithms expect fixed-size observation and action spaces, i.e., a fixed number of state variables and actions. Numeric PDDLGym addresses this by flattening the symbolic states and actions by instantiating all grounded fluents, functions, and actions based on the objects in the given PDDL problem. To motivate RL algorithms to output goal-oriented policies, we treat goal states as terminal states and define a reward function that gives a reward of one for states that achieve the goal and zero otherwise. Moreover, the framework supports the optional encoding of Boolean goals as binary features appended to the observation vector, allowing DRL algorithms to condition their policies on goal information when available.

[6] In the first iteration, M is empty, so the planner cannot run; we treat this as a failed planning attempt.

[7] Technically speaking, our agent uses NSAM * [29], an advanced version of NSAM.

[8] The source code for the Numeric PDDLGym environment is available at: https://github.com/SPL-BGU/NumericPDDLGym

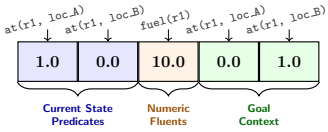

Figure 2: The Numeric PDDLGym observation encoding.

Table 2: Grounded statistics of the domains.

Domain	SMALL		LARGE	
	$\|obs\|$	$\|action\|$	$\|obs\|$	$\|action\|$
Counters	10	12	13	16
Sailing	6	9	12	22
Depot	62	256	80	450
Pogo	150	42	406	106

Gym environments do not support action preconditions; thus, in Numeric PDDLGym, executing an action whose preconditions are violated leaves the state unchanged. Other behaviors are possible, such as terminating the episode with a negative reward. Also, our current conversion mechanism does not support numeric goal conditions and conditional effects. Finally, note that while Numeric PDDLGym supports observation and action spaces of variable sizes, most DRL algorithms do not. Therefore, in our experiments, we limited the number of objects in the problem to a fixed size.

Example. Consider a simple numeric planning problem involving one robot (r_1) and two locations (loc_A, loc_B). The robot's initial state is at loc_A with 10 units of fuel, and its goal is to reach loc_B. The domain defines a single lifted action, `move`, which moves an agent from one location to another and reduces one unit of fuel. Numeric PDDLGym will create two specific operators (for `move`(loc_A, loc_B) and `move`(loc_B, loc_A)). Figure 2 illustrates the translation from the symbolic PDDL state to the numeric observation vector.

6 EXPERIMENTAL SETUP

We compared RAMP against a PPO baseline, equipped with the masking technique to prevent repeating inapplicable actions. As benchmark domains, we considered all the numeric planning domains used by Mordoch et al. [27]. From this set of domains, we used only domains that (1) are supported by Numeric PDDLGym (see its limitations above), (2) have linear preconditions and effects, and (3) have a problem generator. This resulted in three numeric domains: Counters [33], Depot [26], and Sailing [32]. To add diversity, we also experimented with Pogo Stick [3], a recently introduced numeric planning domain based on Polycraft [10], a symbolic wrapper for the popular Minecraft game. This domain involves an agent tasked to craft a wooden pogo, which requires collecting resources of different quantities and performing several crafting recipes. See Benyamin et al. [3] for more details. We improved their original Pogo Stick domain slightly by filtering out some inapplicable actions during problem generation to closely resemble the agent's behavior in Polycraft.

For each domain, we generated problems from two "hardness" levels: *Small* and *Large*. *Large* problems have more objects, resulting in a larger set of grounded fluents, functions, and actions, which are harder to learn and to plan with. Table 2 reports the vector sizes of the observation and action spaces for each domain and hardness level. We generated 50 random problem instances per domain and hardness level with a constant goal, using NSAM's generator for IPC domains [27] and the generator from Benyamin et al. [3] for Minecraft. In each episode, a new problem instance was sampled from this set, and the evaluated algorithm was used to solve it. Each episode ended when either the problem had been solved (i.e., the goal had been reached) or after 1,500 environment steps. When the agent performs an illegal action (i.e., an action that does not satisfy its preconditions), it remains in place.

Configuration and hyperparameters. For planning, we used *Metric-FF* planner [13] with a 60-second time limit. All experiments were repeated five times using different random seeds and conducted on a system with 16 cores of an AMD EPYC 7763 processor and 32GB of RAM. The PPO hyperparameter used by the baseline PPO and RAMP are:learning rate $\alpha = 10^{-3}$, discount factor $\gamma = 0.999$, value-function loss coefficient 0.65, entropy coefficient 0.01, clipping parameter $\epsilon = 0.2$, gradient clipping threshold 1.0, and 3 optimization epochs per update using the Adam optimizer. We used a single environment runner with a batch size matched to the rollout fragment length and equal to 1500. In each of our domains, the problems had a unique goal, and only the initial state was varied. Thus, we omitted encoding the goal in the observation space.

Metrics. We consider two main performance metrics to assess the overall algorithm over the training set: *success rate*, computed as a moving average over the last 25 episodes; and *cumulative solution length*, defined as the total number of steps performed across all successful episodes so far. We truncate results at the point where the success rate flattens, i.e., where further computation no longer yields meaningful improvements. The number of trials conducted per domain is fixed: Counters (100), Sailing (100), Depot (1000), and Pogo Stick (300).

Moreover, we evaluate the correctness of the action model component of RAMP using the predictive power metrics proposed in [39], namely the precision and recall of the preconditions and Boolean effects on a test set.

7 RESULTS

Solvability . First, we consider the success rate results, depicted in Figure 3(left). The success rates of RAMP and PPO are compared across the four domains for the Small and Large hardness levels. As can be seen, RAMP demonstrates superior performance in almost all cases. In Counters and Sailing, RAMP reaches near-perfect solvability much faster than PPO. In Depot, RAMP maintains a clear advantage, solving problems even in the hard instances, where PPO could not solve any of the problems. In these problems, the planning component in RAMP is able to find a solution with the learned domain model in over 90% of cases, leading to higher overall solvability rates. The results for the Pogo Stick domain are less conclusive: in the small hardness level RAMP still outperforms PPO, but for the large hardness level, there is no statistically significant difference. We conjecture that this is because the planner is limited to 60 seconds, and thus it may fail to find a solution even when one exists. Indeed, the success rate drops to around 60% for small Pogo Stick instances and approaches zero for large instances, which leads to RAMP and PPO achieving comparable success rates.

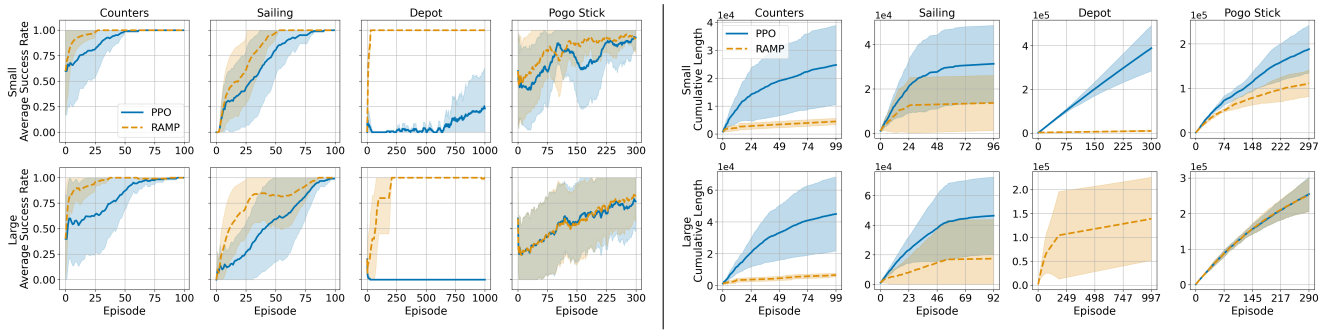

Figure 3: (Left) Rolling average success rate with 95% confidence intervals. (Right) Cumulative solution length with 95% confidence intervals. Top row: Small instances; bottom row: Large instances; columns: different domains.

Plan Quality . Next, consider the *cumulative solution length* results in Figure 3(right). Results are in log scale, where lower values indicate better performance (and shorter plans). For the Large instances in Depot, RAMP was the only agent to solve any instance; therefore, only its results are reported. RAMP consistently finds significantly shorter plans than PPO across both difficulty levels, except in the pogo task, where plan lengths are similar due to the planner frequently failing on large instances, limiting RAMP's ability to optimize the plan. This highlights the benefit of the planner in guiding the RL agent toward more efficient solutions, rather than just finding *any* solution. Even when the planner does not always succeed in finding a plan, its guidance improves the quality and efficiency of the solutions that are found. Although there is a theoretical risk that the RL agent may inherit sub-optimality from an incomplete or suboptimal planner, we did not observe this in practice.

Action Model Quality . To evaluate the action model part of RAMP, we computed the average precision and recall of the learned preconditions and effects against the ground-truth domain models. We ran Metric-FF and 200 random walks on 50 newly generated instances to estimate these metrics. For action effects, RAMP achieves perfect performance, with precision and recall of 1.0 across all evaluated domains. For preconditions, the safety guarantees of NSAM ensure a precision of 1.0.

Recall varies based on the exploration data gathered (Table 3 train columns). Counters achieves near-perfect recall, while Depot remains significantly lower, and Sailing and Pogo Stick fall in between. Compared to the standard offline NSAM trained on an expert dataset of 80 trajectories, the offline approach achieves perfect recall, whereas our method does not. This is because our algorithm prioritizes task solvability over exhaustive recovery of the full action model. For example, in the Depot (small) domain, only 4.2 trajectories on average were sufficient to learn a model capable of solving subsequent problems, indicating that perfect recall is not essential for effective planning.

Plan Efficiency . The DRL policy actively leverages the planner whenever a valid plan exists; across our training instances (as seen in Table 3 test column), RAMP successfully utilizes the planner's plan in over 85% of cases for Counters and Sailing, and in over 93% of the cases for Depot. The planner is only ignored when it fails to find a solution (e.g., timing out on Large Pogo Stick instances), at which point PPO defaults to independent exploration. This ensures that the agent consistently benefits from the high-quality, efficient plans produced by the symbolic model when available, without inheriting the planner's incompleteness.

Table 3: Performance of NSAM in RAMP across domains and sizes. Precondition recall is measured on a held-out test set, while solved and timeout statistics are collected during online training.

		Test	Train	
Domain	**Size**	**Recall**	**Solved**	**Timeout**
counters	small	0.960	97.4	1.2
counters	large	0.990	96.6	2.0
sailing	small	0.696	87.4	0.8
sailing	large	0.849	85.2	7.0
depot	small	0.128	995.8	2.6
depot	large	0.217	931.8	60.4
pogo_stick	small	0.677	169.8	109.8
pogo_stick	large	0.729	0.4	281.8

8 CONCLUSIONS AND FUTURE WORK

In this work, we introduced RAMP, a hybrid strategy for online numeric planning that integrates RL, numeric action model learning (NSAM), and planning. By simultaneously learning a safe domain model and an RL policy, RAMP creates a positive feedback loop: the model generates plans to guide the agent, while the agent's exploration refines the model. Experiments show RAMP significantly outperforms PPO in both solvability and solution length on 3 IPC domains and a recently introduced Minecraft domain.

Moreover, we introduced Numeric PDDLGym, a framework that converts PDDL2.1 domains into Gym environments with fixed-size observation and action spaces. By grounding the symbolic representations and converting them into fixed-length vectors, it enables the direct application of standard RL and DRL methods to numeric planning domains.

Future work will focus on relaxing the assumption of noise-free observability by incorporating probabilistic state representations and noise-robust action model learning, enabling deployment in realistic, partially observable environments.

ACKNOWLEDGMENTS

Yarin Benyamin gratefully acknowledges support from the STEM Scholarship for Outstanding Doctoral Students at the Kreitman School of Advanced Graduate Studies. This research was funded by ISF grants No. 1238/23 to Roni Stern. Additional support was provided by Israel's Ministry of Innovation, Science and Technology (MOST) under Grant No. 1001706842, in collaboration with the Israel National Road Safety Authority and Netivei Israel, awarded to Shahaf Shperberg.

REFERENCES

[1] Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. 2019. Learning action models with minimal observability. *Artificial Intelligence* 275 (2019), 104–137.

[2] Masataro Asai and Alex Fukunaga. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 32.

[3] Yarin Benyamin, Argaman Mordoch, Shahaf Shperberg, Wiktor Piotrowski, and Roni Stern. 2024. Crafting a Pogo Stick in Minecraft with Heuristic Search. In *International Symposium on Combinatorial Search*. 261–262.

[4] Yarin Benyamin, Argaman Mordoch, Shahaf S Shperberg, and Roni Stern. 2025. Toward PDDL Planning Copilot. *arXiv preprint arXiv:2509.12987* (2025).

[5] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dkebiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafał Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Ponda de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[6] Rohan Chitnis, Tom Silver, Joshua B Tenenbaum, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2021. Glib: Efficient exploration for relational model-based reinforcement learning via goal-literal babbling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11782–11791.

[7] Stephen Cresswell, Thomas McCluskey, and Margaret West. 2013. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review* 28, 2 (2013), 195–213.

[8] Maria Fox and Derek Long. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research* 20 (2003), 61–124.

[9] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: theory and practice*.

[10] Stephen A Goss, Robert J Steininger, Dhruv Narayanan, Daniel V Olivença, Yutong Sun, Peng Qiu, Jim Amato, Eberhard O Voit, Walter E Voit, and Eric J Kildebeck. 2023. Polycraft World AI Lab (PAL): An Extensible Platform for Evaluating Artificial Intelligence Agents. *arXiv preprint arXiv:2301.11891* (2023).

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

[12] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

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

[14] Shengyi Huang and Santiago Ontañón. 2022. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. In *FLAIRS*.

[15] Steven James, Benjamin Rosman, and GD Konidaris. 2022. Autonomous learning of object-centric abstractions for high-level planning. In *International Conference on Learning Representations (ICLR)*.

[16] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparametrization with Gumble-Softmax. In *International Conference on Learning Representations (ICLR)*.

[17] Mu Jin, Zhihao Ma, Kebing Jin, Hankz Hankui Zhuo, Chen Chen, and Chao Yu. 2022. Creativity of AI: Automatic symbolic option discovery for facilitating deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 7042–7050.

[18] Brendan Juba, Hai S. Le, and Roni Stern. 2021. Safe Learning of Lifted Action Models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 379–389.

[19] Brendan Juba and Roni Stern. 2022. Learning probably approximately complete and safe action models for stochastic worlds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9795–9804.

[20] Rushang Karia, Pulkit Verma, Gaurav Vipat, and Siddharth Srivastava. 2023. Epistemic Exploration for Generalizable Planning and Learning in Non-Stationary Stochastic Settings. In *NeurIPS 2023 Workshop on Generalization in Planning*.

[21] Leonardo Lamanna, Alessandro Saetti, Luciano Serafini, Alfonso Gerevini, Paolo Traverso, et al. 2021. Online Learning of Action Models for PDDL Planning.. In *IJCAI*. 4112–4118.

[22] Leonardo Lamanna and Luciano Serafini. 2024. Action Model Learning from Noisy Traces: a Probabilistic Approach. In *ICAPS*. AAAI Press, 342–350.

[23] Hai S Le, Brendan Juba, and Roni Stern. 2024. Learning Safe Action Models with Partial Observability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 20159–20167.

[24] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning (ICML)*. https://arxiv.org/pdf/1712.09381

[25] Alan Lindsay, Jonathon Read, Joao F Ferreira, Thomas Hayton, Julie Porteous, and PJ Gregory. 2017. Framer: Planning Models from Natural Language Action Descriptions. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

[26] Derek Long and Maria Fox. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research* 20 (2003), 1–59.

[27] Argaman Mordoch, Brendan Juba, and Roni Stern. 2023. Learning Safe Numeric Action Models. In *AAAI*. AAAI Press, 12079–12086.

[28] Argaman Mordoch, Enrico Scala, Roni Stern, and Brendan Juba. 2024. Safe Learning of PDDL Domains with Conditional Effects. In *ICAPS*. AAAI Press, 387–395.

[29] Argaman Mordoch, Shahaf S. Shperberg, Roni Stern, and Brendan Juba. 2024. Enhancing Numeric-SAM for Learning with Few Observations. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

[30] Jun Hao Alvin Ng and Ronald PA Petrick. 2019. Incremental Learning of Planning Actions in Model-Based Reinforcement Learning. In *IJCAI*. 3195–3201.

[31] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. http://jmlr.org/papers/v22/20-1364.html

[32] Enrico Scala, Patrik Haslum, and Sylvie Thiébaux. 2016. Heuristics for Numeric Planning via Subgoaling. In *IJCAI*. 3228–3234.

[33] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramirez. 2020. Subgoaling techniques for satisficing and optimal numeric planning. *Journal of Artificial Intelligence Research* 68 (2020), 691–752.

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

[35] José Á Segura-Muros, Raúl Pérez, and Juan Fernández-Olivares. 2021. Discovering relational and numerical expressions from plan traces for learning action models. *Applied Intelligence* (2021), 1–17.

[36] Tom Silver and Rohan Chitnis. 2020. PDDLGym: Gym environments from PDDL problems. *arXiv preprint arXiv:2002.06432* (2020).

[37] Sarath Sreedharan and Michael Katz. 2023. Optimistic exploration in reinforcement learning using symbolic model estimates. *Advances in Neural Information Processing Systems* 36 (2023), 34519–34535.

[38] Roni Stern and Brendan Juba. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In *the International Joint Conference on Artificial Intelligence (IJCAI)*. 4405–4411.

[39] Roni Stern, Leonardo Lamanna, Argaman Mordoch, Yarin Benyamin, Pascal Lauer, Brendan Juba, Gregor Behnke, Christian Muise, Pascal Bercher, Mauro Vallati, Kai Xi, Omar Wattad, and Omer Eliyahu. 2025. Evaluating Planning Model Learning Algorithms. In *Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) at ICAPS*.

[40] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[41] Marcus Tantakoun, Christian Muise, and Xiaodan Zhu. 2025. LLMs as planning formalizers: A survey for leveraging large language models to construct automated planning models. In *Findings of the Association for Computational Linguistics: ACL 2025*. 25167–25188.

[42] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. 2024. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032* (2024).

[43] Pulkit Verma, Rushang Karia, and Siddharth Srivastava. 2023. Autonomous capability assessment of sequential decision-making systems in stochastic settings. *Advances in Neural Information Processing Systems* 36 (2023), 54727–54739.

[44] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, P Georgiev, Alexander S Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. 2017. StarCraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782* (2017).

[45] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8 (1992), 279–292.

[46] Christopher John Cornish Hellaby Watkins. 1989. *Learning from delayed rewards*. Ph.D. Dissertation. Oxford: King's College.
[47] Kai Xi, Stephen Gould, and Sylvie Thiébaux. 2024. Neuro-Symbolic Learning of Lifted Action Models from Visual Traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 34. 653–662.
[48] Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, Qiaobo Chen, Yinyuting Yin, Hao Zhang, Tengfei Shi, Liang Wang, Qiang Fu, Wei Yang, and Lanxiao Huang. 2020. Mastering complex control in MOBA games with deep reinforcement learning. In *AAAI*, Vol. 34. 6672–6679. https://doi.org/10.1609/aaai.v34i04.6144
[49] Håkan LS Younes and Michael L Littman. 2004. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162* 2, 99 (2004), 12.