In ICAPS 2023 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2023)
Prague, Czech Republic

Autonomous Capability Assessment of Black-Box Sequential Decision-Making Systems

Pulkit Verma, Rushang Karia, and **Siddharth Srivastava**
Autonomous Agents and Intelligent Robots Lab,
School of Computing and Augmented Intelligence, Arizona State University, USA
{verma.pulkit, Rushang.Karia, siddharths}@asu.edu

Abstract

It is essential for users to understand what their AI systems can and can't do in order to use them safely. However, the problem of enabling users to assess AI systems with evolving sequential decision making (SDM) capabilities is relatively understudied. This paper presents a new approach for modeling the capabilities of black-box AI systems that can plan and act, along with the possible effects and requirements for executing those capabilities in stochastic settings. We present an active-learning approach that can effectively interact with a black-box SDM system and learn an interpretable probabilistic model describing its capabilities. Theoretical analysis of the approach identifies the conditions under which the learning process is guaranteed to converge to the correct model of the agent; empirical evaluations on different agents and simulated scenarios show that this approach is few-shot generalizable and can effectively describe the capabilities of arbitrary black-box SDM agents in a sample-efficient manner.

1 Introduction

AI systems are becoming increasingly complex, and it is becoming difficult even for AI experts to ascertain the limits and capabilities of such systems, as they often use black-box policies for their decision making process (Greydanus et al. 2018; Popov et al. 2017). E.g., consider an elderly couple with a household robot that learns and adapts to their specific household. How would they determine what it can do, what effects their commands would have, and under what conditions? Although we are making steady progress on learning for sequential decision-making (SDM), the problem of enabling users to understand the limits and capabilities of their SDM systems is largely unaddressed. Moreover, as the example above illustrates, the absence of reliable approaches for user-driven capability assessment of AI systems limits their inclusivity and real-world deployability.

This paper presents a new approach for *Query-based Autonomous Capability Estimation* (QACE) of black-box SDM systems in stochastic settings. Our approach uses a restricted form of interaction with the input SDM agent (referred to as SDMA) to learn a probabilistic model of its capabilities. The learned model captures high-level user-interpretable capabilities, such as the conditions under which an autonomous vehicle could back out of a garage, or reach a certain target location, along with the probabilities of possible outcomes of executing each such capability. The resulting learned models directly provide interpretable representations of the scope of SDMA's capabilities. They can also enable and support approaches for explaining SDMA's behavior that require closed-form models (e.g., (Sreedharan, Srivastava, and Kambhampati 2018)). We assume that the input SDMA provides a minimal query-response interface that is already commonly supported by contemporary SDM systems. In particular, SDMA should reveal capability names defining how each of its capabilities can be invoked, and it should be able to accept user-defined instructions in form of sequences of such capabilities. These requirements are typically supported by SDM systems by definition.

The main technical problem for QACE is to automatically compute "queries" in the form of instruction sequences and policies, and to learn a probabilistic model for each capability based on SDMA's "responses" in the form of executions. Depending on the scenario, these executions can be in the real world, or in a simulator for safety-critical settings. Since the set of possible queries of this form is exponential in the state space, naïve approaches for enumerating and selecting useful queries based on the information gain metrics are infeasible.

The main contributions of this work are: (i) the first approach for query-based assessment of SDMAs with minimal assumptions on SDMA internals, and (ii) the first approach to reduce query synthesis for SDMA assessment to FOND planning (Cimatti, Roveri, and Traverso 1998). Empirical evaluation shows that these contributions enable our method to carry out scalable assessment in both embodied and vanilla SDMAs.

We express the learned models using an input concept vocabulary that is known to the target user group. Such vocabularies span multiple tasks and environments. They can be acquired through parallel streams of research on interactive concept acquisition (Kim, Shah, and Doshi-Velez 2015; Kim et al. 2018; Koh et al. 2020; Lage and Doshi-Velez 2020) or explained to users through demonstrations and training (Schulze et al. 2000). These concepts can be modeled as binary-valued *predicates* that have their associated evaluation functions (Mao et al. 2022). We use the syntax and semantics of probabilistic planning domain definition language (PPDDL) (Younes and Littman 2004), to express the learned models.

Figure 1: The cafe server robot environment in OpenRave simulator.

```
(:capability pick-item
 :parameters (?location ?item)
 :precondition (and
   (empty-arm) (has-charge)
   (robot-at ?location)
   (at ?location ?item))
 :effect (and (probabilistic
   0.7 (and (not (empty-arm))
        (not (at ?location ?item))
        (holding ?item))
   0.2 (and (not (has-charge)))
   0.1 (and)))          #No-change
```

Figure 2: Sample PPDDL description for the *pick-item* capability of the cafe server robot.

Related work on the problem addresses model learning from passively collected observations of agent behavior (Pasula, Zettlemoyer, and Kaelbling 2007; Martínez et al. 2016; Juba and Stern 2022); and by exploring the state space using simulators (Ng and Petrick 2019; Chitnis et al. 2021; Mao et al. 2022). However, passive learning approaches can learn incorrect models as they do not have the ability to generate interventional or counterfactual data; exploration techniques can be sample inefficient because they don't take into account uncertainty and incompleteness in the model being learned to guide their exploration. (see Sec. 7 for a greater discussion). In addition to the key contributions mentioned earlier, our results (Sec. 6) show that the approaches for query synthesis in this paper do not place any additional requirements on black-box SDMA but significantly improve (i) convergence rate and sample efficiency for learning relational models of SDMAs with complex capabilities, (ii) few-shot generalizability of learned models to larger environments, and (iii) the accuracy of the learned model w.r.t. the ground truth capabilities for SDMA.

2 Preliminaries

SDMA setup In this work, we work with SDMAs that operate in stochastic and fully observable environments. An SDMA can be represented as a 3-tuple $\langle \mathcal{X}, \mathcal{C}, \mathcal{T} \rangle$, where $\mathcal{X}$ is the environment state space that the SDMA operates in, $\mathcal{C}$ is the set of SDMA's capabilities (capability names, e.g., "place object x at location y" or "arrange table x") that the SDMA can execute, and $\mathcal{T} : \mathcal{X} \times \mathcal{C} \times \mathcal{X} \rightarrow [0, 1]$ is the stochastic black-box transition model determining the effects of SDMA's capabilities on the environment and the probabilities associated with them. Note that the semantics of $\mathcal{C}$ are not known to the user(s) and $\mathcal{X}$ may not be user-interpretable. The only input from the SDMA is the instruction set in the form of capability names, represented as $\mathcal{C}_N$. This isn't a restricting assumption because the AI agents must reveal their instruction sets for usability.

Running Example Consider a cafe server robot that can pick and place items like plates, cans, etc., from various locations in the cafe, like the counter, tables, etc., and also move between these locations. A capability *pick-item (?location ?item)* would allow a user to instruct the robot to pick up an item like a soda can for any location. However, without knowing the capability description, the user would not know under what conditions the robot could execute this capability and what the effects will be.

Object-centric concept representation We aim to learn representations that are generalizable, i.e., the transition dynamics learned should be impervious to the environment properties like number of objects and their configuration. Additionally, the learned dynamics should hold in different settings of objects in the environment as long as the SDMA's capabilites does not change. To this effect, we learn the SDMA's transition model in terms of interpretable concepts that can be represented using first-order logic *predicates*. This is a common formalism used to represent symbolic models for SDMAs (Mao et al. 2022; Zhi-Xuan et al. 2020). We formally represent them using a set of object-centric predicates $\mathcal{P}$. The set of predicates used for cafe server robot in Fig. 1 can be *(empty-arm)*, *(has-charge)*, *(robot-at ?location)*, *(at ?location ?item)*, and *(holding ?item)*. Here, *?* precedes an argument that can be replaced by an object in the environment. E.g., *(robot-at tableRed)* means "robot is at the red table." As mentioned earlier, we assume these predicates along with their boolen evaluation functions (which evaluate to true if predicate is true in a state) are available as input. Learning such predicates (Mao et al. 2022; Sreedharan et al. 2022) is interesting but orthogonal research direction, and it is not the focus of this work.

Abstraction Using an object-centric predicate representation induces an abstraction of environment states $\mathcal{X}$ to high-level logical states $\mathcal{S}$ expressible in predicate vocabulary $\mathcal{P}$. This abstraction can be formalized using a surjective function $f : \mathcal{X} \rightarrow \mathcal{S}$. E.g., in the case of the cafe server robot, the concrete state x may refer to $\langle x, y, z, r, p, \gamma \rangle$ tuples for all objects representing their positions in xyz coordinate with roll, pitch, and yaw values, respectively. On the other hand, the abstract state s corresponding to x will consist of truth values of all the predicates.

Probabilistic transition model Following the framework proposed by Mao et al. (2022), we assume that there exists an arbitrary latent space $\mathcal{S}$ expressible in $\mathcal{P}$. This induces

an abstract transition model $\mathcal{T}' : \mathcal{S} \times \mathcal{C} \times \mathcal{S} \rightarrow [0, 1]$. This is done by converting each transition $\langle x, c, x' \rangle \in \mathcal{T}$ to $\langle s, c, s' \rangle \in \mathcal{T}'$ using predicate evaluators such that $f(x) = s$ and $f(x') = s'$. Now, $\mathcal{T}'$ can be expressed as model M that is a set of parameterized action (capability in our case) schema, where each $c \in \mathcal{C}$ is described as $c = \langle \textit{name(c)}, \textit{pre(c)}, \textit{eff(c)} \rangle$, where *name*$(c) \in \mathcal{C}_N$ refers to name and arguments (parameters) of c; *pre(c)* refers to the preconditions of the capability c represented as a logical formula defined over $\mathcal{P}$ that must be true in a state to execute c; and *eff(c)* refers to the set of logical formulas over $\mathcal{P}$, each of which becomes true on executing c with an associated probability. The result of executing c for a model M is a state $c(s) = s'$ such that $P_M(s'|s, c) > 0$ and one (and only one) of the effects of c becomes true in s'. We also use $\langle s, c, s' \rangle$ triplet to refer to $c(s) = s'$. This representation is similar to the probabilistic planning domain definition language (PPDDL) (Younes and Littman 2004), which can compactly describe the SDMA's capabilities. E.g., the cafe server robot has three capabilities (shown here as `name(args)`): `pick-item(?location ?item)`; `place-item(?location ?item)`; and `move(?source ?destination)`. The description of `pick-item` in PPDDL is shown in Fig. 2.

Variational Distance Given a black-box SDMA $\mathcal{A}$, we learn the probabilistic model M representing its capabilities. To measure how close M is to the true SDMA transition model $\mathcal{T}'$, we use variational distance – a standard measure in probabilistic-model learning literature (Chitnis et al. 2021; Martínez et al. 2016; Ng and Petrick 2019; Pasula, Zettlemoyer, and Kaelbling 2007). It is based on the *total variation distance* between two probability distributions $\mathcal{T}'$ and M, given as:

$$\delta(\mathcal{T}', M) = \frac{1}{|\mathcal{D}|} \sum_{\langle s,c,s' \rangle \in \mathcal{D}} \left| P_{\mathcal{T}'}(s'|s, c) - P_M(s'|s, c) \right| \quad (1)$$

where $\mathcal{D}$ is the set of test samples ($\langle s, c, s' \rangle$ triplets) that we generate using $\mathcal{T}'$ to measure the accuracy of our approach. As shown by Pinsker (1964), $\delta(\mathcal{T}', M) \leq \sqrt{0.5 \times D_{KL}(\mathcal{T}' \,\|\, M)}$, where D_{KL} is the KL divergence.

3 The Capability Assessment Task

In this work, we aim to learn a probabilistic transition model $\mathcal{T}'$ of a black-box SDMA as a model M, given a set of user-interpretable concepts as predicates $\mathcal{P}$ along with their evaluation functions, and the capability names C_N corresponding to the SDMA's capabilities. Formally, the assessment task is:

Definition 1. *Given a set of predicates $\mathcal{P}$ along with their boolean-evaluation functions, capability names $\mathcal{C}_N$, and a black-box SDMA $\mathcal{A}$ in a fully observable, stochastic, and static environment, the* capability assessment task *$\langle \mathcal{A}, \mathcal{P}, \mathcal{C}_N, \mathcal{T}' \rangle$ is defined as the task of learning the probabilistic transition model $\mathcal{T}'$ of $\mathcal{A}$ expressed using $\mathcal{P}$.*

The solution to this task is a model M that should ideally be same as $\mathcal{T}'$ for correctness. In practice, $\mathcal{T}'$ need not be in PPDDL, so the correctness should be evaluated along multiple dimensions.

Notions of model correctness As discussed in Sec. 2, variational distance is one way to capture the correctness of the learned model. This is useful when the learned model and the SDMA's model are not in the same representation. The correctness of a model can also be measured using qualitative properties like soundness and completeness. The learned model M should be sound and complete w.r.t. the SDMA's high-level model $\mathcal{T}'$, i.e., for all combinations of c, s, and s', if a transition $\langle s, c, s' \rangle$ is possible according to $\mathcal{T}'$, then it should also be consistent with M, and vice versa. Here, $\langle s, c, s' \rangle$ is consistent with M if $P(s'|s, c) > 0$ according to M. We formally define this as:

Definition 2. *Let $\langle \mathcal{A}, \mathcal{P}, \mathcal{C}_N, \mathcal{T} \rangle$ be a capability assessment task with a learned model M as its solution. M is* sound *iff each transition $\langle s, c, s' \rangle$ consistent with M is a subset of transitions in $\mathcal{T}'$. M is* complete *iff the set of all transitions in $\mathcal{T}'$ is a subset of transitions consistent with M.*

This also means that if $\mathcal{T}'$ is also a PPDDL model, then (i) any precondition or effect learned as part of M is also present in $\mathcal{T}'$ (soundness), and; (ii) all the preconditions and effects present in $\mathcal{T}'$ should be present in M (completeness). Additionally, a probabilistic model is *correct* if it is sound and complete, and the probabilities for each effect set in each of its capabilities are the same as that of $\mathcal{T}'$.

4 Interactive Capability Assessment

To solve the capability assessment task, we must identify what should be the preconditions and effects of each capability in terms of logical formalae expressed using $\mathcal{P}$. At a very high-level, we do this by identifying that a probabilistic model can be expressed as a set of capabilities $c \in C$, each of which has two places where we can add a predicate p, namely precondition and effect. We call these *locations* within each capability. We then enumerate through these $2 \times |\mathcal{C}|$ locations and figure out how to correctly add the predicate at each of those locations. Here, the correct way to add a predicate to a location can be one of the three ways: (i) adding it as p, i.e., the predicate must be true for that capability to execute (when the location is precondition), or it becomes true on executing it (when the location is effect); (ii) adding it as *not(p)*, i.e., the predicate must be false for that capability to execute (when the location is precondition), or it becomes false on executing it (when the location is effect); (iii) not adding it at all, i.e., the capability execution does not depend on it (when the location is precondition), or the capability does not modify it (when the location is effect).

Hypothesis and Version spaces Let $\mathcal{H}$ represent the hypothesis space of all possible transition models expressible in terms of $\mathcal{P}$ and $\mathcal{C}$. Let $\mathcal{V} \subseteq \mathcal{H}$ represent the version space (Mitchell 1982) corresponding to the set of hypotheses that are consistent with the observed data. In such a setting $\mathcal{T}'$ belongs to $\mathcal{V}$. We must prune the version space to solve the capability assessment task, ideally bringing it to a size of 1. We achieve this by posing queries to the SDMA and using the responses to the queries as data to eliminate the inconsistent hypotheses from the version space.

We generate an exhaustive set of hypotheses for each predicate at every location. Given a location (precondition

or effect in a capability), the hypothesis space corresponding to a predicate will correspond to 3 transition models: one each corresponding to the three ways we can add the predicate in that location. We call these three hypotheses h_T, h_F, h_I, corresponding to adding p (true), *not(p)* (false), and not adding p (ignored), respectively at that location.

Shortening the version spaces Note that the hypothesis space (and also the version space) of the possible transition models is infinite due to the probabilities associated with each transition. To simplify this, we first constrain the hypothesis space by ignoring the probabilities, and hence learning a non-deterministic transition model (commonly referred to as FOND model (Cimatti, Roveri, and Traverso 1998)) instead of a probabilistic one. This makes our hypothesis space finite. We later learn the probabilities using maximum likelihood estimation using the transitions observed as part of the query responses.

Simulator use Using the standard assumption of a simulator's availability in research on SDM, QACE solves the capability assessment task (Sec. 3) by issuing queries to the SDMA and observing its responses in the form of its execution in the simulator. In non-safety-critical scenarios, this approach can work without a simulator too. This interface required to answer the queries is rudimentary as the SDMA $\mathcal{A}$ need not have access to its transition model $\mathcal{T}'$ (or $\mathcal{T}$) but should be able to interact with the environment (or a simulator) to answer the queries. We next present the types of queries we use, followed by algorithms for generating them and inferring the SDMA's model using its responses.

Policy simulation queries (Q_{PS}) These queries ask the SDMA $\mathcal{A}$ to execute a given policy multiple times. More precisely, a Q_{PS} query is a tuple $\langle s_I, \pi, G, \alpha, \eta \rangle$ where $s_I \in \mathcal{S}$ is a state, π is a partial policy that maps each reachable state to a capability, G is a logical predicate formula that expresses a stopping condition, α is an execution cutoff bound, and η is an attempt limit. Note that the query (including the policy) is entirely created by our solution approach without any interaction with the SDMA. Q_{PS} queries ask $\mathcal{A}$ to execute π, η times. In each iteration, execution continues until either the stopping goal condition G or the execution bound α is reached. E.g., "Given that the robot, `soda-can`, `plate1`, `bowl3` are at `table4`, what will happen if the robot follows the following policy: if there is an item on the table and arm is empty, pick up the item; if an item is in the hand and location is not dishwasher, move to the dishwasher; if an item is in the hand and location is dishwasher, place the item in the dishwasher?" Such queries will be used to learn both preconditions and effects (Sec. 4.3). An example of policy simulation queries is included in Appendix A.1.

A response to such queries is an execution in the simulator and η traces of these simulator executions. Formally, the response θ_{PS} for a query $q_{PS} \in Q_{PS}$ is a tuple $\langle b, \zeta \rangle$, where $b \in \{\top, \bot\}$ refers to if the SDMA can reach a goal state $s_G \in G$, and ζ are the corresponding triplets $\langle s, c, s' \rangle$ generated by it when it executed the policy η times. If the SDMA reaches s_G even once during the η simulations, b is $\top$, representing that it is possible to reach the goal using this policy. We next see how we use these responses to prune the version space to learn the correct transition model of the SDMA represented in the input predicate vocabulary.

Algorithm 1: QACE Algorithm

Input : predicates $\mathcal{P}$; capability names $\mathcal{C}_N$; state s; SDMA $\mathcal{A}$; hyperparameters α, η
Output: M

1 $L \leftarrow \{pre, eff\} \times \mathcal{C}_N$
2 $M^* \leftarrow$ initializeModel $(\mathcal{P}, \mathcal{C}_N)$
3 **for** each $\langle l, p \rangle \in \langle L, \mathcal{P} \rangle$ **do**
4 Generate h_T, h_F, h_I by setting p at l in M^*
5 **for** each pair h_i, h_j in $\{h_T, h_F, h_I\}$ **do**
6 $q \leftarrow$ generateQuery$(h_i, h_j, \alpha, \eta, s)$
7 $\theta_{\mathcal{A}}, \mathbb{S} \leftarrow$ getResponse$(q, \mathcal{A}, s)$
8 $M^* \leftarrow$ pruneHypotheses $(\theta_{\mathcal{A}}, h_i, h_j)$
9 $M^* \leftarrow$ learn possible stochastic effects of capability with c_N in l using ζ (in $\theta_{\mathcal{A}}$)
10 $M \leftarrow$ learnProbabilitiesOfStochasticEffects(ζ, M^*)
11 **return** M

4.1 Query-based Autonomous Capability Estimation (QACE) Algorithm

We now discuss how we solve the capability assessment task using the Query-based Autonomous Capability Estimation algorithm (Alg. 1), which works in two phases. In the first phase, QACE learns all preconditions and non-deterministic effects of all the capabilities using the policy simulation queries (Sec. 4.2). In the second phase, QACE converts the non-deterministic effects of capabilities into probabilistic effects (Sec. 4.3). We now explain the learning portion (lines 3-11) of Alg. 1 in detail.

QACE first initializes a model M^* with capabilities having names $c_N \in \mathcal{C}_N$, and predicates $\mathcal{P}$. All the preconditions and effects for all capabilities are empty in this model. QACE iterates over all combinations of L and $\mathcal{P}$ (line 4). For each pair, QACE creates 3 hypotheses h_T, h_F, and h_I as mentioned earlier. It then takes 2 of these (line 5) and generates a query q (line 6) such that the response of the SDMA on that query can help prune out one of the hypotheses (see Sec. 4.2). The query q is then posed to the SDMA $\mathcal{A}$ whose response is stored as $\theta_{\mathcal{A}}$ (line 7). QACE finally prunes at least one of the two hypotheses using $\theta_{\mathcal{A}}$ (line 8). QACE also updates the effects of all models in the version space to fasten the learning process (line 9). Finally, it learns the probabilities of the observed stochastic effects using maximum likelihood estimation (line 10). An important feature of the algorithm (similar to PLEX (Mehta, Tadepalli, and Fern 2011) and AIA (Verma, Marpally, and Srivastava 2021)) is that it keeps track of all the locations where it hasn't identified the correct way of adding a predicate. We next see how QACE generates the queries in line 6.

4.2 Algorithms for Query Synthesis

One of the significant challenges in interactive model learning is to generate the queries we explained above and to learn the agent's model using them. Although active learning (Settles 2012) addresses the related problem of figur-

ing out which data sets to request labels for, vanilla active learning approaches do not apply here because the possible set of queries expressible using the literals in a domain is vast. Query-based learning approaches use an estimate of the value of a query. This can be a multi-valued measure like *information gain* (Sollich and Saad 1994), *value* (Macke, Mirsky, and Stone 2021), etc. or a binary-valued attribute like *distinguishability* (Verma, Marpally, and Srivastava 2021), etc. This is because not all queries are helpful. We use distinguishability as a measure to identify useful queries. According to it, a query q is distinguishing w.r.t. two hypotheses if responses by both models to q do not match. We now discuss methods for generating such queries.

Generating distinguishing queries QACE automates the generation of queries using search. As part of the algorithm, a model M in the version space is used to generate the three hypotheses corresponding to a specific predicate p and location l combination. So other than the predicate p at location l, the model representing the three hypotheses is exactly the same. A forward search is used to generate the policy simulation queries with two hypotheses h_i, h_j chosen randomly from h_T, h_F, and h_I. The forward search is initiated with an initial state $\langle s_0^i, s_0^j \rangle$ as the root of the search tree, where s_0^i and s_0^j are copies of the same state s_0 from which we are starting the search. The edges of the tree correspond to the capabilities with arguments replaced with objects in the environment. The nodes correspond to the two states resulting from applying the capability in the parent state according to the two hypotheses models. E.g., consider that a transition $\langle s_0^i, c, s_1^i \rangle$ is possible according to the model of the hypotheses h_i, and let $\langle s_0^j, c, s_1^j \rangle$ be the corresponding transition (by applying the same effect set of c as h_i) according to the model of the hypotheses h_j. Now there will be an edge in the forward search tree with label c such that parent node is $\langle s_0^i, s_0^j \rangle$ and child node is $\langle s_1^i, s_1^j \rangle$. The search process terminates when a node $\langle s^i, s^j \rangle$ is reached such that either the states s^i and s^j don't match, or the preconditions of the same capability were met in the state according to one of the hypotheses but not according to the other. Forward search can be slow depending on the number of capabilities and objects in the environment. So we use state-of-the-art planner PRP (Muise, McIlraith, and Beck 2012) used for search-based planning in non-deterministic environments. The output of this search is a policy π to reach a state where the two hypotheses, h_i and h_j differs. Additional details about PRP's settings, the input we use for it, and an example of the output policy are available in Appendix A.1. The query $\langle s_I, \pi, G, \alpha, \eta \rangle$ resulting from this search is such that s_I is set to the initial state s_0, π is the output policy, G is the goal state where the hypotheses disagree, α and η are hyperparameters as mentioned earlier. We next see how to use these queries to prune out the incorrect hypothesis.

4.3 Learning Probabilistic Models Using Query Responses

At this point, QACE already has a query such that the response to the query by the two hypotheses does not match. We next see how to prune out the hypothesis inconsistent with the SDMA. QACE poses the query generated earlier to the SDMA and gets its response. If the SDMA can successfully execute the policy, QACE matches the response of the two hypotheses with that of the SDMA and prunes out the hypothesis whose response does not match with that of the SDMA. If the SDMA cannot execute the policy, i.e., SDMA fails to execute some capability in the policy, then the hypotheses cannot be pruned directly. In such a case, a new initial state s_0 must be chosen to generate a new query starting from that initial state. This process to generate new queries for the same pair of hypotheses can take a long time, hence we preempt this issue by creating a pool of states $\mathbb{S}$ that can execute the capabilities using a directed exploration of the state space using partially learned models.

Learning probabilities of stochastic effects After QACE learns the non-deterministic model, to learn the probabilities of the learned effects it uses the transitions collected as part of responses to queries. This is done using Maximum Likelihood Estimation (MLE) (Fisher 1922). For each triplet $\langle s, c, s' \rangle$ seen in the collected data, let $count_c$ be the number of times a capability c is observed. Now, for each effect set, the probability of that effect set becoming true on executing that capability c is given as the number of times that effect is observed on executing c divided by $count_c$. As we increase the value of the hyperparameter η, we increase the number of collected triplets, thereby improving the probability values calculated using this approach.

5 Theoretical Analysis and Correctness

We now discuss how the model M of SDMA $\mathcal{A}$ learned using QACE fulfills the notions of correctness (Sec. 3) discussed earlier. We first show that the model M^* learned before line 10 of QACE (Alg. 1) is sound and complete according to Def. 2. The proofs for the theorems are available in Appendix B.

Theorem 1. *Let $\mathcal{A}$ be a black-box SDMA with a ground truth transition model $\mathcal{T}'$ expressible in terms of predicates $\mathcal{P}$ and a set of capabilities $\mathcal{C}$. Let M^* be the non-deterministic model expressed in terms of predicates $\mathcal{P}^*$ and capabilities $\mathcal{C}$, and learned using the query-based autonomous capability estimation algorithm (Alg. 1) just before line 10. Let C_N be a set of capability names corresponding to capabilities $\mathcal{C}$. If $\mathcal{P}^* \subseteq \mathcal{P}$, then the model M^* is* sound *w.r.t. the SDMA transition model $\mathcal{T}'$. Additionally, if $\mathcal{P}^* = \mathcal{P}$, then the model M^* is* complete *w.r.t. the SDMA transition model $\mathcal{T}'$.*

Next, we show that the final step to learn the probabilities for all the effects in each capability converges to the correct probability distribution of the source distribution under the assumption that all the effects of a capability are identifiable. When a capability c is executed in the environment, one of its effects $e_i(c) \in \mathit{eff}(c)$ will be observed in the environment. To learn the correct probability distribution in M, we should accurately identify that effect $e_i(c)$. Hence, the set of effects is *identifiable* if at least one state exists in the environment from which each effect can be uniquely identified when the capability is executed. An example of this is available in Appendix. A.4.

Theorem 2. *Let $\mathcal{A}$ be a black-box SDMA with a ground truth transition model $\mathcal{T}'$ expressible in terms of predicates $\mathcal{P}$ and a set of capabilities $\mathcal{C}$. Let M be the probabilistic model expressed in terms of predicates $\mathcal{P}^*$ and capabilities $\mathcal{C}$, and learned using the query-based autonomous capability estimation algorithm (Alg. 1). Let $\mathcal{P} = \mathcal{P}^*$ and M be generated using a sound and complete non-deterministic model M^* in line 11 of Alg. 1, and let all effects of each capability $c \in \mathcal{C}$ be identifiable. The model M is* correct *w.r.t. the model $\mathcal{T}'$ in the limit as η tends to ∞, where η is hyperparameter in query Q_{PS} used in Alg. 1.*

6 Empirical Evaluation

We implemented Alg. 1 in Python to evaluate our approach empirically. We found that our query synthesis and interactive learning process leads to (i) few shot generalization; (ii) convergence to a sound and complete model; and (iii) much greater sample efficiency and accuracy for learning lifted SDM models with complex capabilities as compared to the baseline.

SDMAs for evaluation To test the efficacy of our approach, we created SDMAs for five different settings; *Cafe Server Robot* is a Fetch robot (Wise et al. 2016) that can do sequential decision-making in a restaurant environment to serve food, clear tables, etc.; *Warehouse Robot* is a robot that can stack, unstack, and manage the boxes in a warehouse; *Driving Agent* that can drive between locations and can repair the vehicle at certain locations; *First Responder Robot* that can assist in emergency scenarios by driving to emergency spots, providing first-aid and water to victims, etc.; and *Elevator Control Agent* that can control the operation of multiple elevators in a building. Here, the Cafe Server Robot uses the ATM-MDP task and motion planning system (Shah et al. 2020) internally that is unknown to the QACE algorithm, whereas the other four SDM systems use state-of-the-art stochastic planning systems from the literature. Additional details about each setting are available in Appendix C.

Setup We used a *single* training problem with few objects (≤ 7) for all methods in our evaluation and used a test set that was composed of problems containing object counts larger than those in the training set. We ran the experiments on a cluster of Intel Xeon E5-2680 v4 CPUs with CentOS 7.9 running at 2.4 GHz with a memory limit of 8 GB and a time limit of 4 hours. For QACE, we used $\alpha = 2d$ where d is the PRP policy depth and $\eta = 5$. All of the methods in our empirical evaluation receive the same training and test sets and are evaluated on the same platform.

Cafe Server Robot This SDMA setup uses an 8 degrees of freedom Fetch (Wise et al. 2016) robot in a cafe setting on OpenRave simulator (Diankov and Kuffner 2008). The low-level environment state consists of continuous x, y, z, roll, pitch, and yaw values of all objects in the environment. The predicate evaluators were provided by ATM-MDP of which we used only a subset to learn a PPDDL model. Each robot capability is refined into motion controls at run-time depending on the configuration of the objects in the environment. The results for variational distance between the learned model and the ground truth model in Fig. 4 show that despite the different vocabulary, QACE learns an accurate transition model for the SDMA. The baseline was not compatible with ATM-MDP setup hence it was compared with the other four vanilla SDMA settings only.

Baseline Selection We used the closest SOTA related work, GLIB (Chitnis et al. 2021) as a baseline. It learns a probabilistic model of an intrinsically motivated agent by sampling goals far away from the initial state and making the agent try to reach them. This can be adapted to an assessment setting by moving goal-generation based sampling outside the agent, and, to the best of our knowledge, no existing approach addresses the problem of creating intelligent questions for an SDMA. GLIB has two versions, GLIB-G, which learns the model as a set of grounded noisy deictic rules (NDRs) (Pasula, Zettlemoyer, and Kaelbling 2007), and GLIB-L, which learns the model as a set of lifted NDRs. We used the same hyperparameters as published for the *Warehouse Robot* and *Driving Agent* and performed extensive tuning for the others and report results with the best performing settings.

Accuracy To compare accuracy, we use Variational Distance (VD) as presented in Eq. 1. However, GLIB cannot use this measure because it learns a set of NDRs and hence does not have a unique NDR for each capability. In order to maintain parity in comparison, we use GLIB's setup to calculate an approximation of the VD. Using it, we sample 3500 random transitions $\langle s, c, s' \rangle$ from the ground truth transition model $\mathcal{T}'$ using problems in the test set to compute a dataset of transitions $\mathcal{D}$. The sample-based, approximate VD is then given as: $\frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \mathbf{1}_{[s' \neq c_M(s)]}$, where $c_M(s)$ samples the transition using the capability in the learned model output by each method. In Fig. 5, we compare the approximate variational distance of the three approaches w.r.t. $\mathcal{D}$ as we increase the learning time. Note that we also evaluated VD for QACE using Eq. 1 and found that $\delta(\mathcal{T}', M) \approx 0$ for our learned model M in all SDMA settings. The plots in Fig. 5 show the exact point (marked as $\times$) when $\delta(\mathcal{T}', M) \approx 0$. The detailed results are included in Appendix D.

Faster convergence The time taken for QACE to learn the final model is much lower than that of GLIB for three of the four SDMAs. This is because trace collection by QACE is more directed and hence ends up learning the correct model in *a shorter time*. The only setup where GLIB marginally outperforms QACE is Warehouse Robot, and this happens because this SDMA has just two capabilities, one of which is deterministic. Hence, GLIB can easily learn their configuration from a few observed traces. For SDMAs with complex and much larger number of capabilities – First Responder Robot and Elevator Control Agent – GLIB finds it more challenging to learn the model that is closer to the ground truth transition model. Additionally, QACE takes much fewer samples to learn the model than GLIB. In all settings, QACE is much more *sample efficient* than GLIB as QACE needed at most 4% of the samples needed by GLIB-G to reach the variational distance that GLIB-G plateaued at. In contrast, GLIB-L started timing out only after processing a few samples for complex SDMAs.

Few-shot generalization To ensure that learned models are not overfitted, our test set contains problems with larger

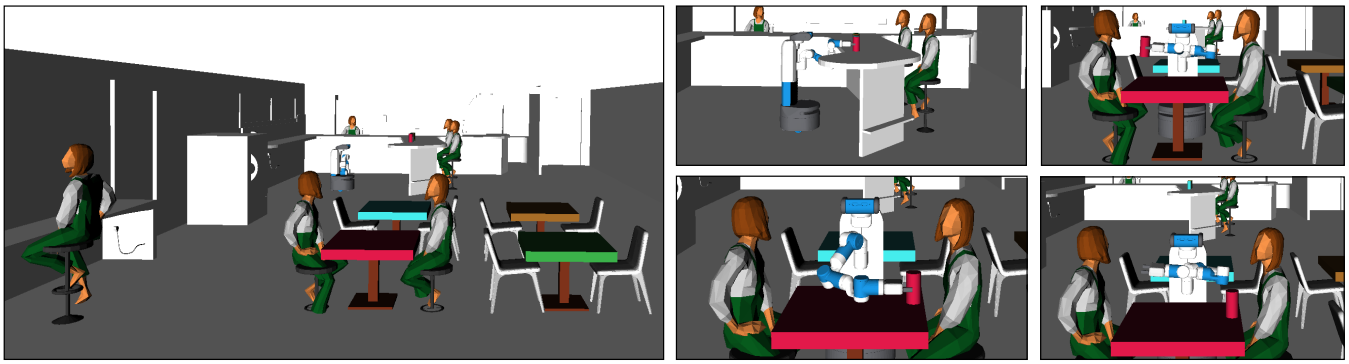

Figure 3: Screen captures from the Cafe Server Robot simulation. The complete environment is shown in the image on the left. The image grid on the right shows screen captures of multiple steps of the robot delivering a `soda-can` to a table.

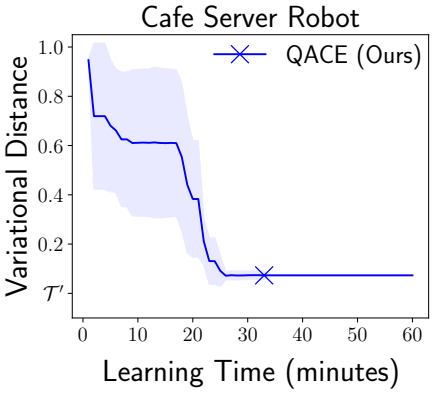

Figure 4: Change in Variational Distance (y) with increasing time (x) for QACE for Cafe Server Robot.

quantities of objects than those used during training. As seen in Fig. 5, GLIB has higher variational distance from the ground truth model for complex domains as compared to QACE . This shows that QACE has *better few-shot generalization* as compared to GLIB.

7 Related Work

The problem of learning probabilistic relational agent models from a given set of observations has been well studied (Juba and Stern 2022; Martínez et al. 2016; Mourão et al. 2012; Pasula, Zettlemoyer, and Kaelbling 2007). Jiménez et al. (2012) and Arora et al. (2018) present comprehensive reviews of such approaches. We next discuss the closest related research directions.

Passive learning Several methods learn a probabilistic model of the agent and environment from a given set of agent executions. Pasula, Zettlemoyer, and Kaelbling (2007) learn the models in the form of noisy deictic rules (NDRs) where an action can correspond to multiple NDRs and also model noise. Mourão et al. (2012) learn such operators using action classifiers to predict the effects of an action. Rodrigues, Gérard, and Rouveirol (2011) learn non-deterministic models as a collection of rule sets and learn these rule sets incrementally. They take a bound on the number of rules as input. Juba and Stern (2022) provide a theoretical framework to learn safe probabilistic models with a range of probabilities for each probabilistic effect while assuming that each effect is atomic and independent of others. A common issue with such approaches is that they are susceptible to incorrect and sometimes inefficient model learning as they cannot control the input data used for learning or perform interventions on it.

Sampling of transitions Several approaches (Jin et al. 2022; Ng and Petrick 2019)) learn the operator descriptions from exploring the state space but focus on deterministic models. The process of evaluating deterministic models is significantly easier and the number of works there would be too broad to cover here. A few reinforcement learning approaches have been explored for learning the relational probabilistic action model by exploring the state space using pre-determined criteria to generate better samples (Ng and Petrick 2019). Konidaris, Kaelbling, and Lozano-Pérez (2018) explore learning PPDDL models for planning, but they aim to learn the high-level symbols needed to describe a set of input low-level options, and these symbols are not interpretable. GLIB (Chitnis et al. 2021) also learns probabilistic relational models using goal sampling as a heuristic for generating relevant data, whereas we use active querying using guided forward search for this. Our empirical analysis shows that our approach to the synthesis of queries yield greater sample efficiency and correctness profiles than the goal generation used in this approach.

Active learning Inspired from Angluin (1988), there are several active learning approaches (Aarts et al. 2012; Tang et al. 2013; Vaandrager 2017) that learn automata to represent the system's model. These approaches assume access to a teacher (or an oracle) that can determine whether the learned automaton is correct and provide a counterexample if it is incorrect. This is not possible in the black-box SDMA settings we work with.

8 Conclusion

In this work, we presented an approach to learning a probabilistic model of an agent using interactive querying. We

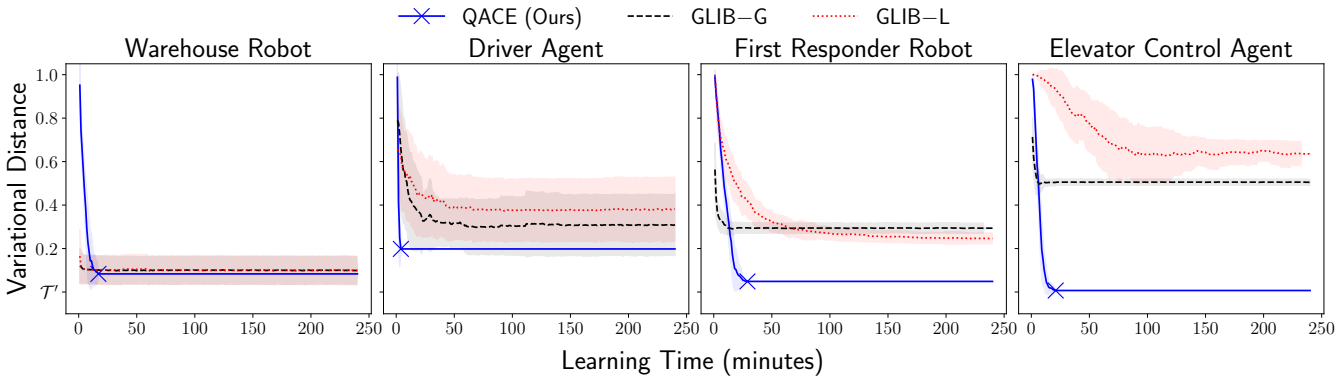

Figure 5: A comparison of the approximate variational distance as a factor of the learning time for the three methods: QACE (ours), GLIB-G, and GLIB-L (lower values better). $\times$ shows that the learning process ended at that time instance for QACE . The results were calculated using 30 runs per method per domain. Solid lines are averages across runs, and shaded portions show the standard deviation. $\mathcal{T}'$ is the ground truth model. Detailed results are available in Appendix D.

show that the approach is few-shot generalizable to larger environments and learns a sound and complete model faster than state-of-the-art approaches in sample-efficient manner.

Limitations and Future Work In this work, we assume that the agent can be connected to a simulator to answer the queries. In real-world settings, sometimes this assumption can be limiting as users might not have direct access to such a simulator. For future work, we must formalize under what conditions is it safe to ask the queries directly to the agent in the real-world instead of passing them to the simulator. Additionally, in this work, we assume the availability of the instruction set of the SDMA as input in the form of capability names. In certain settings, it might be useful to discover the capabilities of an evolving AI SDM system. In future work, methods such as iCaML (Verma, Marpally, and Srivastava 2022) can be used to address this limitation. We also plan to leverage knowledge of queries that the agent fails to answer using approaches like DAAISy (Nayyar, Verma, and Srivastava 2022) that also use negative examples to learn action models efficiently. In the future, we can also extend QACE to work with systems like JEDAI (Shah et al. 2022) as interfaces to make AI systems compliant with Level II assistive AI (Srivastava 2021). Finally, we also plan to do a more rigorous analysis of the complexities of the queries (Verma and Srivastava 2021) used in this paper.

Acknowledgements

We thank Jayesh Nagpal for his help with setting up the Cafe Server Robot SDMA. This work was supported by the ONR under grants N00014-21-1-2045 and N00014-23-1-2416.

References

Aarts, F.; Heidarian, F.; Kuppens, H.; Olsen, P.; and Vaandrager, F. 2012. Automata Learning through Counterexample Guided Abstraction Refinement. In *Proc. ISFM*.

Angluin, D. 1988. Queries and Concept Learning. *Machine Learning*, 2(4): 319–342.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A Review of Learning Planning Action Models. *Knw. Engg. Rev.*, 33: E20.

Bonet, B.; and Givan, R. 2005. 5th International Planning Competition: Non-Deterministic Track.

Bryce, D.; and Buffet, O. 2008. 6th International Planning Competition: Uncertainty Part. In *Proceedings of the 6th International Planning Competition*.

Chitnis, R.; Silver, T.; Tenenbaum, J.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. GLIB: Efficient Exploration for Relational Model-Based Reinforcement Learning via Goal-Literal Babbling. In *Proc. AAAI*.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Strong Planning in Non-Deterministic Domains via Model Checking. In *International Conference on Planning Systems*.

Diankov, R.; and Kuffner, J. 2008. OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical Report CMU-RI-TR-08-34, CMU, Pittsburgh, PA, USA.

Fisher, R. A. 1922. On the Mathematical Foundations of Theoretical Statistics. *Philosophical Transactions of the Royal Society of London, Series A*, 222(594-604): 309–368.

Greydanus, S.; Koul, A.; Dodge, J.; and Fern, A. 2018. Visualizing and Understanding Atari Agents. In *Proc. ICML*.

Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A Review of Machine Learning for Automated Planning. *Knw. Engg. Rev*, 27(4): 433–467.

Jin, M.; Ma, Z.; Jin, K.; Zhuo, H. H.; Chen, C.; and Yu, C. 2022. Creativity of AI: Automatic Symbolic Option Discovery for Facilitating Deep Reinforcement Learning. In *Proc. AAAI*.

Juba, B.; and Stern, R. 2022. Learning Probably Approximately Complete and Safe Action Models for Stochastic Worlds. In *Proc. AAAI*.

Kiefer, J.; and Wolfowitz, J. 1956. Consistency of the Maximum Likelihood Estimator in the Presence of Infinitely Many Incidental Parameters. *The Annals of Mathematical Statistics*, 887–906.

Kim, B.; Shah, J. A.; and Doshi-Velez, F. 2015. Mind the Gap: A Generative Approach to Interpretable Feature Selection and Extraction. In *Proc. NeurIPS*.

Kim, B.; Wattenberg, M.; Gilmer, J.; Cai, C.; Wexler, J.; Viegas, F.; and Sayres, R. 2018. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV). In *Proc. ICML*.

Koh, P. W.; Nguyen, T.; Tang, Y. S.; Mussmann, S.; Pierson, E.; Kim, B.; and Liang, P. 2020. Concept Bottleneck Models. In *Proc. ICML*.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Pérez, T. 2018. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61(1): 215–289.

Lage, I.; and Doshi-Velez, F. 2020. Learning Interpretable Concept-Based Models with Human Feedback. In *ICML 2020 Workshop on Human Interpretability in Machine Learning*.

Macke, W.; Mirsky, R.; and Stone, P. 2021. Expected Value of Communication for Planning in Ad Hoc Teamwork. In *Proc. AAAI*.

Mao, J.; Lozano-Pérez, T.; Tenenbaum, J. B.; and Kaelbing, L. P. 2022. PDSketch: Integrated Domain Programming, Learning, and Planning. In *Proc. NeurIPS*.

Martínez, D.; Alenyà, G.; Torras, C.; Ribeiro, T.; and Inoue, K. 2016. Learning Relational Dynamics of Stochastic Domains for Planning. In *Proc. ICAPS*.

Mehta, N.; Tadepalli, P.; and Fern, A. 2011. Autonomous Learning of Action Models for Planning. In *Proc. NeurIPS*.

Mitchell, T. M. 1982. Generalization as Search. *Artificial Intelligence*, 18(2): 203–226.

Mourão, K.; Zettlemoyer, L.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. In *Proc. UAI*.

Muise, C.; McIlraith, S.; and Beck, C. 2012. Improved Non-Deterministic Planning by Exploiting State Relevance. In *Proc. ICAPS*.

Nayyar, R. K.; Verma, P.; and Srivastava, S. 2022. Differential Assessment of Black-Box AI Agents. In *Proc. AAAI*.

Ng, J. H. A.; and Petrick, R. P. A. 2019. Incremental Learning of Planning Actions in Model-Based Reinforcement Learning. In *Proc. IJCAI*.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning Symbolic Models of Stochastic Domains. *Journal of Artificial Intelligence Research*, 29: 309–352.

Pinsker, M. S. 1964. *Info. and Info. Stability of Random Variables and Processes*.

Popov, I.; Heess, N.; Lillicrap, T.; Hafner, R.; Barth-Maron, G.; Vecerik, M.; Lampe, T.; Tassa, Y.; Erez, T.; and Riedmiller, M. 2017. Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. *arXiv preprint arXiv:1704.03073*.

Rodrigues, C.; Gérard, P.; and Rouveirol, C. 2011. Incremental Learning of Relational Action Models in Noisy Environments. In *Proc. ILP*.

Schulze, K. G.; Shelby, R. N.; Treacy, D. J.; Wintersgill, M. C.; VanLehn, K.; and Gertner, A. 2000. Andes: An Active Learning, Intelligent Tutoring System for Newtonian Physics. *Themes in Education*, 1(2): 115–136.

Settles, B. 2012. *Active Learning*. Morgan & Claypool Publishers. ISBN 1608457257.

Shah, N.; Kala Vasudevan, D.; Kumar, K.; Kamojjhala, P.; and Srivastava, S. 2020. Anytime Integrated Task and Motion Policies for Stochastic Environments. In *Proc. ICRA*.

Shah, N.; Verma, P.; Angle, T.; and Srivastava, S. 2022. JEDAI: A System for Skill-Aligned Explainable Robot Planning. In *Proc. AAMAS*.

Sollich, P.; and Saad, D. 1994. Learning from Queries for Maximum Information Gain in Imperfectly Learnable Problems. In *Proc. NeurIPS*.

Sreedharan, S.; Soni, U.; Verma, M.; Srivastava, S.; and Kambhampati, S. 2022. Bridging the Gap: Providing Post-Hoc Symbolic Explanations for Sequential Decision-Making Problems with Inscrutable Representations. In *Proc. ICLR*.

Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2018. Hierarchical Expertise Level Modeling for User Specific Contrastive Explanations. In *Proc. IJCAI*.

Srivastava, S. 2021. Unifying Principles and Metrics for Safe and Assistive AI. In *Proc. AAAI*.

Tang, Y.; Pacharoen, W.; Aoki, T.; Bhattarakosol, P.; and Surarerks, A. 2013. Active Learning of Nondeterministic Finite State Machines. *Mathematical Problems in Engineering*, 373265.

Vaandrager, F. 2017. Model Learning. *Communications of the ACM*, 60(2): 86–95.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the Right Questions: Learning Interpretable Action Models Through Query Answering. In *Proc. AAAI*.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2022. Discovering User-Interpretable Capabilities of Black-Box Planning Agents. In *Proc. KR*.

Verma, P.; and Srivastava, S. 2021. Learning Causal Models of Autonomous Agents using Interventions. In *IJCAI 2021 Workshop on Generalization in Planning*.

Wise, M.; Ferguson, M.; King, D.; Diehr, E.; and Dymesich, D. 2016. Fetch and Freight: Standard Platforms for Service Robot Applications. In *IJCAI 2016 Workshop on Autonomous Mobile Service Robots*.

Younes, H. L. S.; and Littman, M. L. 2004. PPDDL 1.0: An Extension to PDDL for Expressing Domains with Probabilistic Effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The First Probabilistic Track of the International Planning Competition. *Journal of Artificial Intelligence Research*, 24: 851–887.

Zhi-Xuan, T.; Mann, J.; Silver, T.; Tenenbaum, J.; and Mansinghka, V. 2020. Online Bayesian Goal Inference for Boundedly-Rational Planning Agents. In *Proc. NeurIPS*.

A Additional Details

A.1 Example of Policy Simulation Query

As mentioned in the main paper, these queries ask the SDMA $\mathcal{A}$ to execute a given policy multiple times. More precisely, a Q_{PS} query is a tuple $\langle s_I, \pi, G, \alpha, \eta \rangle$ where $s_I \in \mathcal{S}$ is a state, π is a partial policy that maps each reachable state to a capability, G is a logical predicate formula that expresses a stopping condition, α is an execution cutoff bound, and η is an attempt limit. Note that the query (including the policy) is entirely created by our solution approach without any interaction with the SDMA. Q_{PS} queries ask $\mathcal{A}$ to execute π, η times. In each iteration, execution continues until either the stopping goal condition G or the execution bound α is reached. E.g., "Given that the robot, `soda-can`, `plate1`, `bowl3` are at `table4`, what will happen if the robot follows the following policy: if there is an item on the table and arm is empty, pick up the item; if an item is in the hand and location is not dishwasher, move to the dishwasher?". Fig. 6(right) shows an example of such a query. Note that the initial state is shown adjacent to the top-most node. Note that the initial state can be obtained by abstracting a low level state using predicate evaluators. Fig. 6(left) shows an example of such an abstraction where a low-level state in terms of objects' location is abstracted in terms of user-interpretable predicates.

Such queries can be generated using non-deterministic planners like PRP (Muise, McIlraith, and Beck 2012). How we use PRP for our search process outlined in Sec. 4.2 in the main paper is explained next.

A.2 Generating Queries using PRP

QACE automates the generation of queries using non-deterministic planning problems. QACE always generates queries to distinguish between models that differ only on one predicate corresponding to just one location (a precondition or effect in a capability). To generate the policy simulation queries, QACE creates a FOND planning model and a problem. Let M_i and M_j be a pair of FOND models expressed using $\mathcal{P}$ and $\mathcal{C}$ corresponding to hypotheses h_i and h_j, where $i, j \in \{T, F, I\}$. QACE renames the predicates and capabilities in M_i and M_j as $\mathcal{P}_i$ and $\mathcal{P}_j$, and $\mathcal{C}_i$ and $\mathcal{C}_j$, respectively, so that there are no intersections and a pair of states in the two models can be progressed independently using pairs of capabilities. This gives a planning model M_{ij} expressed in terms of $\mathcal{P}_{ij}$ and $\mathcal{C}_{ij}$. Here, $\mathcal{P}_{ij} = \mathcal{P}_i \cup \mathcal{P}_j \cup \{(goal)\}$, where *(goal)* is a 0-ary predicate. It is used to identify when the goal for the FOND planning problem is reached. For each capability $\langle c_i, c_j \rangle \in \langle \mathcal{C}_i, \mathcal{C}_j \rangle$ such that their names match, $pre(c_{ij})$ of the combined capability c_{ij} is disjunction of preconditions of c_i and c_j. For $e(c_{ij}) \in \mathit{eff}(c_{ij})$ ACE adds three conditional effects: (i) $pre(c_i) \wedge pre(c_j) \Rightarrow e(c_i) \wedge e(c_j)$; (ii) $pre(c_i) \wedge \neg pre(c_j) \Rightarrow (goal)$; and (iii) $\neg pre(c_i) \wedge pre(c_j) \Rightarrow (goal)$. An example of this process is included in the next section.

Starting from an initial state, the FOND problem uses one of these states and maintains two different copies of all the objects in the environment, one corresponding to each of the models. Each model only manipulates the objects in its own copy. QACE then solves a planning problem that has an initial state $s_{I_{ij}} = \{p_i^{*_1}, \ldots, p_i^{*_z}, p_j^{*_1}, \ldots, p_j^{*_z}\}$ and a goal state $G_{ij} = (goal) \vee [\exists p \in \mathcal{P}^*_{ij}(p_i \wedge \neg p_j) \vee (\neg p_i \wedge p_j)]$. Here, $\mathcal{P}^*$ represents the grounded version of predicates $\mathcal{P}$ using objects O in the environment. The partial policy π generated as a solution to this planning problem is a *strong solution*. As shown by Cimatti, Roveri, and Traverso (1998), the solution is a *strong solution* if the resulting plan is guaranteed to reach the goal. The solution partial policy will lead the two models in a state where at least one capability cannot be applied, and hence the *(goal)* predicate becomes true. This is possible because the models differ only in the way one predicate is added at a location. We formalize this with the following lemma. The proof is available in Sec. B.

Lemma 1. *Given two models M_i and M_j such that both are abstractions of the same FOND model, and are at the same level of abstraction with only one predicate differing in way it is added in one of the location, the intermediate FOND planning problem created using QACE to generate policy simulation queries has a strong solution.*

A.3 Planning Problem to Generate Queries

This section provides an example of a sample planning problem using which we generate a query. We provide below an example of how to modify the precondition and effect of a capability when we are learning its preconditions. The procedure is similar for effects.

Consider we have a capability *move (?frm ?to)* in the Cafe server robot, and we already know one of its preconditions; *(has-charge)*. We are now trying to find what will be the correct way to add the predicate *(robot-at ?frm)* in the precondition of this *move (?frm ?to)* capability. Consider we have two models M_i and M_j, where $i = T$ and $j = F$. We will represent their *move* capability as follows:

```
(:action move_i
  :parameters (?frm - loc ?to - loc)
  :precondition (and (has-charge_i)
    (robot-at_i ?frm))
  :effect (and )
)

(:action move_j
  :parameters (?frm - loc ?to - loc)
  :precondition (and (has-charge_j)
    (not (robot-at_i ?frm)))
  :effect (and )
)
```

To create a query, we will combine the *move* capabilitys into a combined capability. For each capability $\langle c_i, c_j \rangle \in \langle \mathcal{C}_i, \mathcal{C}_j \rangle$ s.t. $name(c_i) = name(c_j)$, $pre(c_{ij}) = pre(c_i) \vee pre(c_j)$; and for each $e(c_{ij}) \in \mathit{eff}(c_{ij})$ we add three conditional effects: (i) $pre(c_i) \wedge pre(c_j) \Rightarrow e(c_i) \wedge e(c_j)$; (ii) $pre(c_i) \wedge \neg pre(c_j) \Rightarrow (goal)$; and (iii) $\neg pre(c_i) \wedge pre(c_j) \Rightarrow (goal)$. Applying it here for the *move* capability, we get:

```
(:action move_ij
  :parameters (?frm - loc ?to - loc)
  :precondition (or
    (and (has-charge_i)
```

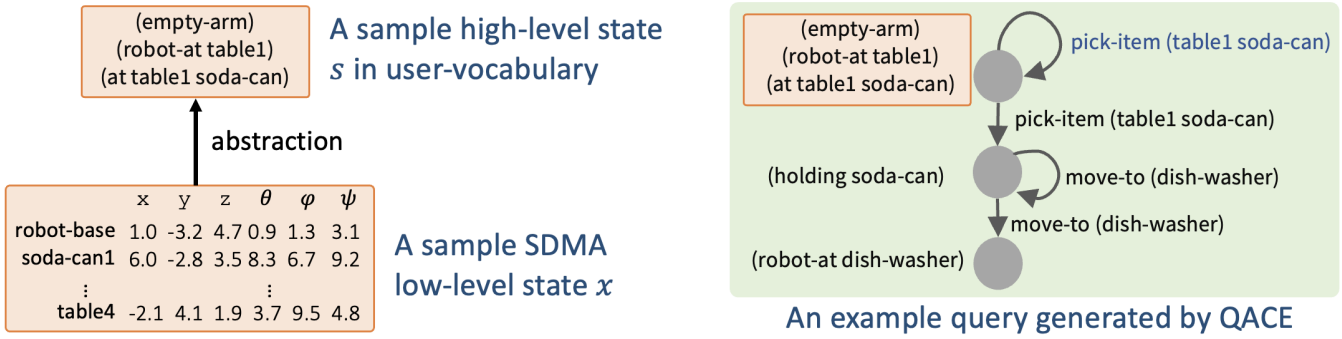

Figure 6: An example of abstraction of low-level state into a high level state (left) and an example of a policy simulation query (right). For the policy, the labels on the left of nodes correspond to state properties that must be true in those states, and the labels on right of edges correspond to the capabilitys for each eadge. The policy simulation query corresponds to: "Given that the robot and `soda-can` are at `table1`, what will happen if the robot follows the following policy: if there is an item on the table and arm is empty, pick up the item; if an item is in the hand and location is not dishwasher, move to the dishwasher?".

```
        (robot-at_i ?frm))
      (and (has-charge_j)
        (not (robot-at_j ?frm)))
    )
    :effect (and
      (when (and (has-charge_i)
          (robot-at_i ?frm)
          (has-charge_j)
          (not (robot-at_j ?frm)))
        (and )
      )
      (when (and (has-charge_i)
          (robot-at_i ?frm)
          (or (not (has-charge_j))
            (robot-at_j ?frm)))
        (and (goal))
      )
      (when (and (has-charge_j)
          (not (robot-at_j ?frm))
          (or (not (has-charge_i))
            (not (robot-at_i ?frm))))
        (and (goal))
      )
    )
)
```

Note that we have expanded $pre(c_i) \wedge \neg pre(c_j)$ using disjunction of negations of all predicates in $pre(c_j)$, etc.

A.4 Identifiable Effects

A set of effects of a capability are identifiable if there exists a state such that when we execute a capability in that state, we can identify which of its effects was executed. Let us consider a capability a, such that $pre(a) = \{p_1 \wedge p_2 \wedge \neg p_3\}$, and $eff(a) = \{\langle p_3 \wedge p_4, 0.2\rangle, \langle p_3 \wedge \neg p_2, 0.5\rangle, \langle p_3 \wedge \neg p_4 \wedge \neg p_2, 0.3\rangle\}$. The effects of this capability are identifiable because if we execute this capability in state $\{p1, p2, p4\}$, we can identify which of its effect is getting executed. This is because, on executing a, we can identify each effect as follows: (i) if the resulting state has p_4 and p_2, then it is the first effect, (ii) if the resulting state has p_4 but not p_2, then it is the second effect, and (iii) if the resulting has neither p_2 nor p_4, then it is the third effect.

A.5 Instantiated Predicates

A literal corresponding to a predicate $p \in \mathcal{P}$ can appear in $pre(c)$ or any $e_i(c) \in eff(c)$ of a capability $c \in \mathcal{C}$ iff it can be instantiated using a subset of parameters of c. E.g., consider a capability *move (?src ?dest)* and a predicate *(connected ?x ?y)* in the example discussed earlier. Suppose a literal corresponding to the predicate *(connected ?x ?y)* can appear in the precondition and/or the effect of *move (?src ?dest)*. The possible lifted instantiations of predicate *connected* compatible with *move-car* are *(connected ?src ?dest)*, *(connected ?dest ?src)*, *(connected ?src ?src)*, and *(connected ?dest ?dest)*. The number of parameters in a predicate $p \in P$ that is relevant to a capability $c \in \mathcal{C}$, i.e., instantiated using a subset of parameters of c, is bounded by the maximum arity of c. So using the capability names and the predicates, we get a set of instantiated predicates. In our implementation we use these set of instantiated predicates as the set of predicates.

B Theoretical Results

We will next show that the plan in the distinguishing queries always ends up with the capability that is part of the pal tuple being concretized at that time. This will help us in limiting our analysis to, at most, the last 2 capabilities in the plan.

Proposition 1. *Let M_i, M_j be the models corresponding to hypotheses h_i, h_j, where $i, j \in \{T, F, I\}$. These models generated by adding a predicate p in a location corresponding to a capability c to a model M. Suppose $q = \langle s_I, \pi, G, \alpha, \eta\rangle$ is a distinguishing query for two distinct models M_i, M_j. The last capability in the partial policy π to achieve G will be c.*

Proof. We prove this by contradiction. Consider that the last capability of the policy π in the distinguishing query q is $c' \neq c$. Now the query q used to distinguish between

M_i and M_j is generated using the FOND planning problem $\langle M_{ij}, s_{I_{ij}}, G_{ij} \rangle$, which has a solution if both the models have different precondition or at least one different effect for the same capability. Since the last capability of the policy is c', the two models either have different preconditions for c' or different effects. This is not possible as, according to Alg. 1, M_i and M_j differ only in precondition or effect of one capability c. Hence $c' = c$. □

We now use this proposition to prove Lemma 1 stated in Appendix A.2.

Lemma 2. *Given two models M_i and M_j such that both are abstractions of the same FOND model, and are at the same level of abstraction with only one predicate differing in way it is added in one of the location, the intermediate FOND planning problem created using QACE to generate policy simulation queries has a strong solution.*

Proof (Sketch). We prove this in two parts. In the first part, we consider the case where we are refining the model in terms of the precondition of some capability. Recall that for each capability c_{ij}, we have 3 conditional effects: i) $pre(c_i) \wedge pre(c_j) \Rightarrow e(c_i) \wedge e(c_j)$; (ii) $pre(c_i) \wedge \neg pre(c_j) \Rightarrow$ *(goal)*; and (iii) $\neg pre(c_i) \wedge pre(c_j) \Rightarrow$ *(goal)*. Now, according to proposition 1, capability c_{ij} has to be the last capability in the policy π. Since the model M_i and M_j differ only in preconditions, condition (ii) or (iii) must be true for c_{ij}. This implies that on executing c_{ij}, the $(goal)$ predicate will become true, and executing this policy π will end up in reaching the goal.

In the second part, we consider the case where we are refining the model in terms of the effects of some capability. According to proposition 1, capability c_{ij} has to be the last capability in the policy π. Since the model M_i and M_j differ only in effects, condition (i) must be true for c_{ij}. This implies that on executing c_{ij}, one of the predicates will become true according to one model, and false according to another, and hence executing this policy π will end up in reaching the goal condition G_{ij}. □

Next, we prove the soundness and completeness of the learned model w.r.t. the agent model. Note that an important part of the process is to get a state s, where a capability c can be executed successfully. We can collect this information using some random traces, using a state where all capabilities are applicable, or asking the agent for a state where certain conditions are met (Q_{SR}). We use this information in the proof.

Theorem 1. *Let $\mathcal{A}$ be a black-box SDMA with a ground truth transition model $\mathcal{T}'$ expressible in terms of predicates $\mathcal{P}$ and a set of capabilities $\mathcal{C}$. Let M^* be the nondeterministic model expressed in terms of predicates $\mathcal{P}^*$ and capabilities $\mathcal{C}$, and learned using the query-based autonomous capability estimation algorithm (Alg. 1) just before line 10. Let $\mathcal{C}_N$ be a set of capability names corresponding to capabilities $\mathcal{C}$. If $\mathcal{P}^* \subseteq \mathcal{P}$, then the model M^* is* sound *w.r.t. the SDMA transition model $\mathcal{T}'$. Additionally, if $\mathcal{P}^* = \mathcal{P}$, then the model M^* is* complete *w.r.t. the SDMA transition model $\mathcal{T}'$.*

Proof. We first prove that given the predicates P, capability names $\mathcal{C}_H$, model of the agent $\mathcal{T}'$, and the model M^* learned by ALg. 1, M^* is sound w.r.t. the model $\mathcal{T}'$. We do this in two cases. The first one showing that the learned preconditions of all the capabilitys in M^* are sound, and the second one showing the same thing for learned effects. We use M_T, M_F, and M_I to refer to models corresponding to hypotheses h_T, h_F, and h_I, respectively.

Case 1: Consider the location is precondition in a capability c where we are trying to find the correct way to add a predicate $p \in \mathcal{P}$.
Case 1.1: Let the models we are comparing be M_T and M_I (or M_F). The policy simulation query q to distinguish between these models would involve executing c in a state where p is false. Now, M_T would fail to execute c (as it has p as a positive precondition), and M_I (or M_F) would successfully execute it. If $\mathcal{A}$ can execute c in such a state, we can filter out the model M_T. We can also remove p from a state where $\mathcal{A}$ is known to execute c, and see if it can execute c. If not, we can filter out the model M_I (or M_F).
Case 1.2: Let the models we are comparing be M_F and M_I. The policy simulation query q to distinguish between these models would involve executing c in a state where p is true. M_F would fail to execute c as it has p as a negative precondition, whereas M_I would successfully execute it. If $\mathcal{A}$ can execute c in such a state, we can filter out the model M_T. We can also add p to a state where $\mathcal{A}$ is known to execute c, and see if it can execute c. If not, we can filter out the model M_I.

Case 2: Consider the location is effect in a capability c where we are trying to find the correct way to add a predicate $p \in \mathcal{P}^*$.
Case 2.1: Let the models we are comparing be M_T and M_I (or M_F). The policy simulation query q used to distinguish between these models would involve executing c in a state where p is false. After executing it, the resulting state will have p true according to M_T only. We ask the agent to simulate the policy N times, with p as the goal formula G. If p appears in any of the simulation after executing c, then we learn all the possible effects involving p. Not that the capability has identifiable effects, so if p appears in more than one effect, the corresponding effect will eventually be discovered when concretizing the predicate that uniquely identifies that effect.
Case 2.2: Let the models we are comparing be M_F and M_I. The policy simulation query q used to distinguish between these models would involve executing c in a state where p is true. After executing it, the resulting state will have p true according to M_I only. We ask the agent to simulate the policy η times, with p as the goal formula G. If p appears in any of the runs, then we learn all the possible effects involving p. Not that the capability has identifiable effects, so if p appears in more than one effect, the corresponding effect will eventually be discovered when concretizing the predicate that uniquely identifies that effect.

Combining both cases, we infer that whenever we learn a precondition or effect, it is added in the same form as in

the ground truth model $\mathcal{T}'$, hence the learned model M^* is sound w.r.t. $\mathcal{T}'$.

We now prove that given the predicates $\mathcal{P}$, capability names $\mathcal{C}_H$, model of the agent $\mathcal{T}'$, and the model M^* learned by ALg. 1, M^* is complete w.r.t. the model $\mathcal{T}'$. We just showed that the model that we learn is sound as whenever we add a predicate in a precondition or effect, it is in correct mode. Now, since Alg. 1 loops over all possible combinations of predicates and capabilities, for both precondition and effect, we will learn all the preconditions and effects correctly. Hence, the learned model will be complete w.r.t. the agent model. □

Theorem 2. *Let $\mathcal{A}$ be a black-box SDMA with a ground truth transition model $\mathcal{T}'$ expressible in terms of predicates $\mathcal{P}$ and a set of capabilities $\mathcal{C}$. Let M be the probabilistic model expressed in terms of predicates $\mathcal{P}^*$ and capabilities $\mathcal{C}$, and learned using the query-based autonomous capability estimation algorithm (Alg. 1). Let $\mathcal{P} = \mathcal{P}^*$ and M be generated using a sound and complete non-deterministic model M^* in line 11 of Alg. 1, and let all effects of each capability $c \in \mathcal{C}$ be identifiable. The model M is* correct *w.r.t. the model $\mathcal{T}'$ in the limit as η tends to ∞, where η is hyperparameter in query Q_{PS} used in Alg. 1.*

Proof (Sketch). Thm. 1 showed that the model learned by Alg. 1 is sound and complete, meaning all the preconditions and effects are correctly learned. Consider that each sample generated by asking an agent to follow a policy is i.i.d. Now, if we consider only the samples in which a capability is applied in a state such that its effects are identifiable effects, then we can use MLE to learn the correct probabilities given infinite such samples. This is a direct consequence of the result that given infinite i.i.d. samples, probabilities learned by MLE converge to the true probabilities (Kiefer and Wolfowitz 1956). □

C SDMA Setups – Additional Information

We used five SDMA setups for our experiments. As stated in the main paper, we used a single, small training problem with few objects (≤ 7). To demonstrate generalizability, our test set contained problems that had twice the number of objects than the training problem. Increasing the number of objects causes an exponential increase in the problem size in terms of the state space. Short descriptions of each SDMA setup is presented below:

Cafe Server Robot This SDMA setup uses 8 degrees of freedom Fetch (Wise et al. 2016) robot in a cafe setting on OpenRave simulator (Diankov and Kuffner 2008). As shown in Fig. 6 (left), the low-level environment state consists of continuous x, y, z, roll, pitch, and yaw values of all objects in the environment. The predicate evaluators were provided by ATM-MDP (Shah et al. 2020) of which we used only a subset to learn a PPDDL model. Each robot capability is refined into motion controls at run-time depending on the configuration of the objects in the environment.

Warehouse Robot This SDM setup is implemented using state-of-the-art stochastic planning system used in planning literature. This is motivated from *Exploding Blocksworld* setup introduced in the probabilistic track of International Planning Competition (IPC) 2004 (Younes et al. 2005). It features a robot that has four capabilities: stack, unstack, pick, and place. stack capability stacks one object on top of another, unstack capability removes an object from top of another object, pick capability picks up an object from a fixed location, where place capability places the object at a fixed location. The setup is non-deterministic as executing some of these capabilities can destroy the object as they might be delicate. Hence even the ground truth does not have 100% success rate in this setup.

Driver Agent This SDM setup is implemented using state-of-the-art stochastic planning system used in planning literature. This is motivated from *Tireworld* setup introduced in the probabilistic track of IPC 2004 (Younes et al. 2005). It consists of a robot moving around multiple locations. The move action between locations can cause it to get a flat-tire with some probability. Not all locations have the option to change tire, but if available, a change-tire action will fix the flat-tire with a 100% probability.

First Responder Robot This SDM setup is inspired from *First Responders* in uncertainty track of IPC 2008 (Bryce and Buffet 2008). The setup features two kinds of emergencies: fire and medical, involving hurt victims. Victims can be treated at the site of an emergency or the hospital. This was originally a FOND setup, and we added probabilities to all the capabilities with non-deterministic effects to make it probabilistic. The recovery status depending on the treatment location, is different with different probabilities.

Elevator Control Agent This SDM setup is motivated from *Elevators* in the probabilistic track of IPC 2006 (Bonet and Givan 2005). It consists of an agent managing multiple elevators on multiple floors in a single building. The capabilities of moving from one elevator to another on the same floor are probabilistic. The size of this setup is much larger than the previous three. Also, the capabilities have arities of up to 5, making this setup complex from an assessment point of view.

D Extended Empirical Evaluation

In addition to the experiments described in the main paper, we also performed some additional experiments. The results for the same are illustrated in Fig. 8 and Fig. 7.

Results w.r.t. Environment Steps Fig. 7 show a comparison of the approximate variational distance between QACE and the baselines as a factor of the total steps taken in the environment. From the results, it is clear that QACE is able to outperform GLIB while taking far fewer steps in the environment. GLIB-L operates by babbling lifted goals and we found that the goal babbling step of GLIB-L took an inordinate amount of time leading to very few steps in the environment before the timeout of 4 hours. GLIB-G babbles grounded goals and thus can perform many steps but is not sample efficient in learning as the results show. We analyzed the cause and found that if GLIB-G learns an incorrect model, it is often quite difficult to get out of local minima since it keeps generating and following the same plan.

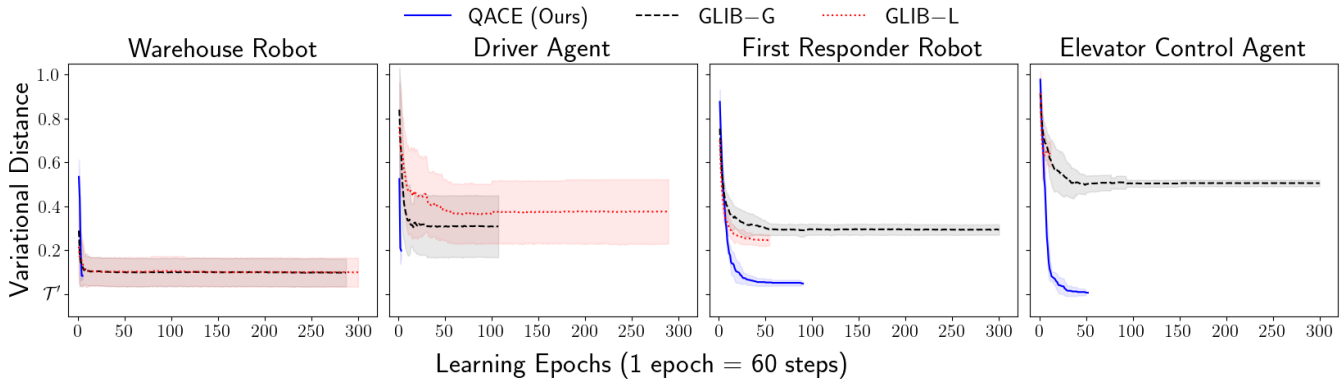

Figure 7: Results showing the trends in the approximate Variational Distance w.r.t. the total number of steps in the environment (lower values better) for the three methods: QACE (ours), GLIB-G, and GLIB-L. Lines which do not extend until the end indicate that the time limit (4 hours) was exceeded. The results were calculated using 30 runs per method per domain. Solid lines are averages across runs, and shaded portions show the standard deviation. $\mathcal{T}'$ is the ground truth model.

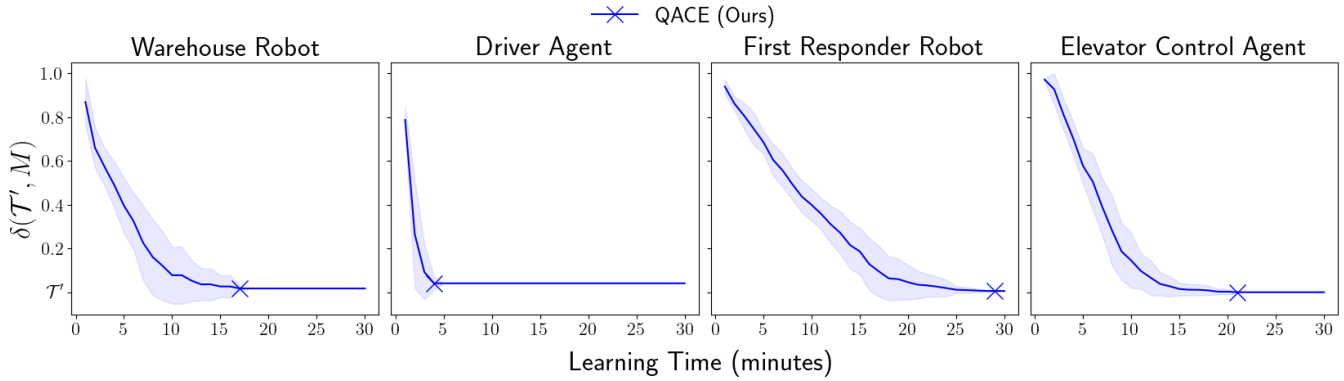

Figure 8: Results showing the comparison of QACE w.r.t. the ground truth model $\mathcal{T}'$. The plots show a trend in the variational distance (see Eq. 1) as a factor of the learning time for QACE (lower values better). $\times$ shows that the learning process ended at that time instance for QACE. The results were calculated using 30 runs per method per domain. Solid lines are averages across runs, and shaded portions show the standard deviation.

Evaluation w.r.t. Ground Truth Models $\mathcal{T}'$ Fig. 8 demonstrate that QACE is able to converge to a learned model that is near-perfect compared to the ground truth model $\mathcal{T}'$. QACE is able to learn such a near-perfect model in a fraction of the time compared to the baselines (see Fig. 5 of the main paper). QACE can learn the non-deterministic effects and preconditions in a finite number of representative environment interactions and given enough samples MLE estimates are guaranteed to converge. This is in stark contrast to GLIB whose learned NDRs cannot be easily compared to the ground truth.