Robotics: Science and Systems 2021
Held Virtually, July 12–16, 2021

Active Learning of Abstract Plan Feasibility

Michael Noseworthy*, Caris Moses*, Isaiah Brand*, Sebastian Castro,
Leslie Kaelbling, Tomás Lozano-Pérez, Nicholas Roy
MIT, CSAIL

***Abstract*—Long horizon sequential manipulation tasks are effectively addressed hierarchically: at a high level of abstraction the planner searches over abstract action sequences, and when a plan is found, lower level motion plans are generated. Such a strategy hinges on the ability to reliably predict that a feasible low level plan will be found which satisfies the abstract plan. However, computing *Abstract Plan Feasibility* (APF) is difficult because the outcome of a plan depends on real-world phenomena that are difficult to model, such as noise in estimation and execution. In this work, we present an active learning approach to efficiently acquire an APF predictor through task-independent, curious exploration on a robot. The robot identifies plans whose outcomes would be informative about APF, executes those plans, and learns from their successes or failures. Critically, we leverage an *infeasible subsequence property* to prune candidate plans in the active learning strategy, allowing our system to learn from less data. We evaluate our strategy in simulation and on a real Franka Emika Panda robot with integrated perception, experimentation, planning, and execution. In a stacking domain where objects have non-uniform mass distributions, we show that our system permits real robot learning of an APF model in four hundred self-supervised interactions, and that our learned model can be used effectively in multiple downstream tasks[1].**

I. Introduction

Long horizon sequential manipulation tasks still pose a challenging problem for robotic systems. Tasks such as assembly depend on using many objects with varying physical properties. Finding a plan to achieve a task in these domains consists of reasoning over large spaces that include discrete action plans, as well as low level continuous motion plans.

These problems can be effectively addressed hierarchically: at the highest level of abstraction the system searches over plausible *abstract* action sequences, and at the lower level it plans for detailed *concrete* motion plans and object interactions. The complexity of the search space for the concrete planner is greatly reduced when constrained by the abstract action sequence. Further computational efficiencies can be gained if we lazily [17] postpone concrete planning until we have a complete abstract plan that is likely to succeed, avoiding the need to query the concrete planner multiple times. A version of this approach is used in *skeleton-based* task and motion planning systems [25, 16, 22].

The success of this lazy strategy hinges on our ability to predict whether an abstract action sequence will be feasible to execute. In this work, we call an abstract action sequence feasible if both the concrete planner returns a solution and this solution is reliably executed in the real world with the intended outcome. If the abstract planner does not take into account errors in execution due to phenomena unmodeled by the concrete planner, the robot may end up attempting a plan that fails during execution.

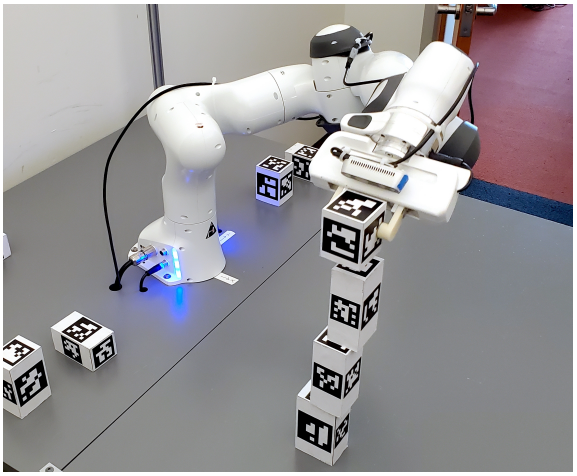


Fig. 1: The Franka Emika Panda robot constructing a tower to improve its understanding of plan feasibility. A wrist-mounted camera refines object pose estimates for precise grasping.

Fortunately, even an approximately correct estimator of *abstract plan feasibility* (APF) can offer huge computational advantages during planning. In some cases it may be possible to approximate APF via coarse-grained simulation. However, this strategy still requires a coarse dynamics model, which may not capture complex phenomena needed to accurately predict feasibility in the real world.

Instead, we explore a strategy in which we learn a model that predicts APF by exploring the space of real plan executions without a specific planning problem or task at hand — a form of curious exploration [27]. Data efficiency is a primary concern in enabling real robot learning of feasibility models. Here, a training instance is the execution trace of an abstract plan, labeled by success or failure. Labeling each such plan is very expensive as it involves finding and executing a concrete motion plan, potentially taking several minutes on a real robot. Furthermore, due to the combinatorial input space of abstract action plans, randomly executing actions is unlikely to elicit interesting behavior.

To address the data efficiency problem, we observe that in the process of training the APF model, some observations may be more valuable than others. Active learning is a technique for identifying unlabeled instances that are most informative in learning a target concept. The technical challenge is how to

[1]An accompanying video can be found at https://youtu.be/UF-SjGm20Mw
*Equal contribution.

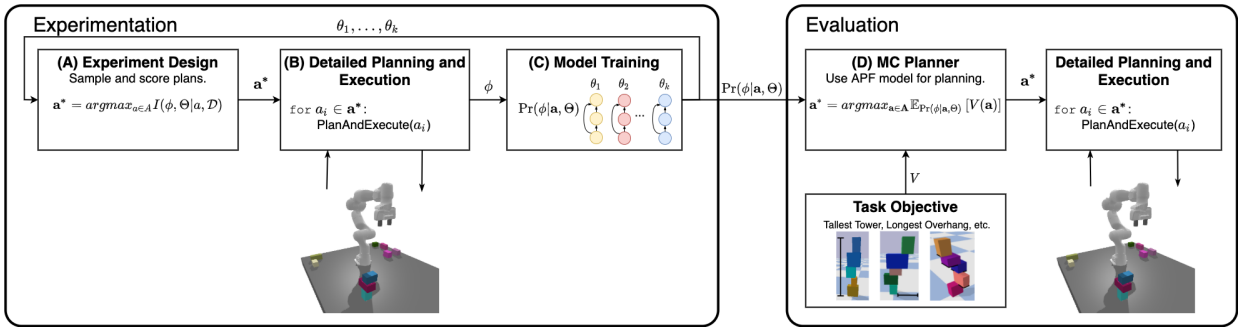


Fig. 2: The proposed system for learning *Abstract Plan Feasibility* (APF) operates in two phases. **Experimentation Phase** (left) The robot iteratively designs and executes experiments that improve its APF model. **(A)** Using its current model, the robot selects the abstract action sequence, $\mathbf{a}^*$, that minimizes its entropy over the APF model. **(B)** The robot then computes and executes a concrete motion plan for $\mathbf{a}^*$. **(C)** After observing the true plan feasibility, ϕ, the robot uses this new labeled data to update its APF model, represented as an ensemble of neural networks. **Evaluation Phase** (right) Once an APF model has been learned, the robot can use it to perform various tasks, such as building the tower with the longest overhang from a given set of blocks.

find plans of interest — an active learning approach requires both a way to generate candidate plans, and a way to score how informative a candidate plan might be given the current model.

To determine how informative a plan is with respect to the learned APF model, we adopt an information-theoretic active learning approach [26, 19]. To generate candidate plans, we exploit an important property of abstract action sequences: for an action sequence $(a_1, \ldots, a_n)$, if any prefix $(a_1, \ldots, a_i)$ is infeasible, then any longer prefix $(a_1, \ldots, a_j)$ for $i < j \leq n$ is also infeasible. This *infeasible subsequence property* gives us leverage during data acquisition. A complex plan instance may contain many elements that are highly informative for model learning, but will never be experienced because early elements in the plan will fail with high probability.

We apply this active learning strategy to the concrete problem of stacking blocks with a real robot, where the blocks are each unique and have non-uniform mass distributions. The robot autonomously designs, plans, and executes experiments to learn a feasibility model using a Franka Emika Panda robot arm (Figure 1). The robot is also capable of resetting the world state after each experiment, enabling continuous autonomous experimentation. The learned feasibility predictor is later used to build towers with previously unseen blocks that satisfy several different objective functions, including the tallest possible tower or the tower with the longest overhang. This sample-efficient autonomous learning process relieves engineers from supervising data collection, resetting the experimental environment, and having to specify accurate dynamics models for planning. This results in a highly flexible and robust system for planning and executing complex action sequences in the real world.

In summary, our contributions are:

- A method to learn an *Abstract Plan Feasibility* model by synthesizing hypothetical plans;
- A data acquisition approach which leverages the *infeasible subsequence property* when sampling potential plans;
- A robotic system which conducts autonomous self-supervised learning via integrated perception, experimentation, planning, and execution.

II. Problem Formulation

Our objective is to learn a model that predicts the success of an abstract action sequence when it is executed by the robot. That is, to learn the parameters Θ that predict

$$\Pr(\phi \mid \mathbf{a}; \Theta),$$

where $\phi \in \{0, 1\}$ is the success of the sequence of abstract actions, $\mathbf{a} = (a_1, \ldots, a_n)$. Furthermore, we wish to learn Θ using as few labeled action sequences $(\mathbf{a}, \phi)$ as possible.

Note that in general, APF may need to consider the initial state in which a plan is to be executed, however in this work we make the assumption that the first abstract action in a sequence will be feasible, and does not depend on the initial state. The method applies regardless of whether this assumption is made.

To learn this APF model, our system operates in two phases as illustrated in Figure 2. In the *experimentation phase*, the robot curiously explores the space of possible plans, learning the parameters of the APF model; in the *evaluation phase*, the robot is given specific goals to achieve, and uses the learned APF model to efficiently plan over abstract action sequences.

Both phases depend on the ability to construct and execute concrete plans on a real robot given an abstract action sequence. We assume that a system is available that can integrate perception, planning, and control to execute an abstract action sequence and observe whether or not it was successful. In Section IV we describe our implementation of this system in detail.

A. Experimentation phase

During the experimentation phase, the robot performs active learning to efficiently design, plan, and execute abstract action sequences that are informative about APF. This phase operates in a loop. At each iteration, the robot first generates candidate action sequences that may be informative according to some *sampling strategy*. Each action sequence is then scored using the current model according to some acquisition function, $f : \mathbf{A} \to \mathbb{R}$. The highest scoring plans are executed on the robot

to obtain feasibility labels which are then used to update the model for future iterations of the active learning loop. The effectiveness of active learning in reducing data complexity depends critically upon the choice of acquisition function and sampling strategy — or which action sequences are considered as possible experiments. These choices are discussed in more detail in Section III.

B. Evaluation phase

After the APF model has been learned, it can be used to achieve multiple objectives. We assume that the robot is given an objective function V which maps action sequences to real values and its goal is to find the abstract action sequence, $\mathbf{a} \in \mathbf{A}$ with the highest expected value,

$$\mathbf{a}^* = \underset{\mathbf{a} \in \mathbf{A}}{\operatorname{argmax}} \, \mathbb{E}_{\Pr(\phi|\mathbf{a},\Theta)} \left[V(\mathbf{a}) \right] \tag{1}$$

where the expectation takes into account the likelihood that an abstract action sequence is feasible when executed on the real robot. To maximize this objective, we use a Monte Carlo planner which randomly samples action sequences and selects the one with the maximum expected value.

III. Active Learning of Abstract Plan Feasibility

Collecting data on real robot platforms is both time and cost-intensive. To minimize the amount of data needed to learn the APF model, we take an information theoretic approach to active learning [19]. Concretely, we maintain a distribution over model parameters that are consistent with the data we have observed so far. Our objective is to select new data that minimize the entropy over this distribution as quickly as possible. Efficient active learning requires: (1) a model class that captures uncertainty in model parameters, (2) a way to score unlabeled plans based on how informative they may be, and (3) a method of generating potentially informative plans. We discuss each of these in turn.

A. Abstract plan feasibility model

Our APF model, $\Pr(\phi \mid \mathbf{a}; \Theta)$, aims to capture the uncertainty in the underlying stochastic process of predicting the feasibility of abstract action sequences. This uncertainty can be attributed to phenomena such as the robot's motor capabilities, errors in perception, or unmodeled behaviors of the planning process, and is referred to as *aleatoric uncertainty*. Our goal is to learn parameters Θ such that this uncertainty is adequately captured and our model can be leveraged, along with a low level planner, to achieve a goal.

We take a Bayesian approach to learning the model parameters, and maintain a distribution over the parameter space, $\Pr(\Theta)$. This distribution aims to capture the uncertainty we have regarding the accuracy of our predictions, referred to as *epistemic uncertainty*. In general, for complex model classes such as neural network classifiers, an explicit representation of $\Pr(\Theta \mid \mathcal{D})$ for training data $\mathcal{D}$ is difficult to construct or update with new data. We therefore follow the strategy of Beluch et al. [3] and represent this uncertainty with an ensemble of N models, $(\theta_1, \ldots, \theta_N)$, where $\theta_i \in \mathbb{R}^d$. Initial parameters are drawn independently at random and are updated to incorporate new data via gradient descent.

The design of the models in the ensemble is selected to match the underlying structure of the prediction problem. In a naive implementation, we can directly estimate feasibility from the entire sequence of actions. We will refer to this approach as a *complete* model, denoted by Θ_{comp}. However, we observe that the *infeasible sub-sequence property* provides a strong constraint on the set of feasible plans: sequence $(a_1, \ldots, a_n)$ is only feasible if all of its prefixes are also feasible. As such, the model can instead learn the probability a specific action is feasible given all previous actions were feasible. We will refer to a model that considers this property as Θ_{ss}. Under this model, the feasibility of a plan, $\mathbf{a}$, is:

$$\Pr(\Phi_{1:n} \mid \mathbf{a}, \Theta_{ss}) = \prod_{i=2}^{n} \Pr(\Phi_i \mid a_{1:i}, \Phi_{1:i-1} = \mathbf{1}; \Theta_{ss}), \tag{2}$$

where Φ_i represents whether action a_i is feasible given $a_{1:i-1}$ were feasible, and we use the subscript $1:n$ to refer to a sequence of variables. In our method we assume initial actions are feasible, meaning $\Pr(\Phi_1 = 1 \mid a_1) = 1$.

The Θ_{comp} model only considers full action sequences, and therefore entire plans correspond to a single label once they are executed. On the other hand, Θ_{ss} requires labels after each action is executed.

In both of these models, the length of plans for which we require predictions varies. Therefore, we use *graph neural networks* (GNNs), which make predictions based on aggregations of local properties and relations among the input entities, and exploit parameter tying to model global properties of plans of arbitrary size using a fixed-dimensional parameterization Θ. For a detailed description of GNNs, see the overview by [39]. More details about the specific architecture we use can be found in Section IV-C.

B. Entropy reduction

Following [26, 6], we guide our active learning by picking a sequence of data $\mathcal{D}$ that maximally reduces the entropy of $\Pr(\Theta \mid \mathcal{D})$. The general problem of designing a sequence of experiments to minimize entropy — or equivalently, maximize information gain — is a difficult sequential decision-making problem. Fortunately, due to sub-modularity of the objective, a myopic approach that considers only the next experiment to conduct can be shown to be a good approximation to the optimal experimentation strategy [9].

Given a model distribution $\Pr(\Theta \mid \mathcal{D})$ that depends on the data we have seen so far, $\mathcal{D}$, we choose the action sequence $\mathbf{a} \in \mathbf{A}$ that reduces the entropy of the posterior distribution as much as possible:

$$\mathbf{a}^* = \underset{\mathbf{a} \in \mathbf{A}}{\operatorname{argmax}} \, \mathrm{H}(\Theta \mid \mathcal{D}) - \mathbb{E}_{\phi \sim \Pr(\cdot \mid \mathcal{D}, \mathbf{a})} \left[\mathrm{H}(\Theta \mid \mathcal{D}, \mathbf{a}, \phi) \right] \tag{3}$$

While the robot can select the plan, $\mathbf{a}$, to experiment with, it cannot select the outcome ϕ, so to compute this quantity, we have to take an expectation over the outcome, using our current model distribution.

Estimating the entropy over a high-dimensional parameter space is expensive, so we follow the approach of Houlsby et al. [19] to reformulate the objective in (3) as:

$$\mathbf{a}^* = \underset{\mathbf{a}\in\mathbf{A}}{\operatorname{argmax}}\ \mathrm{I}(\Phi : \Theta \mid \mathcal{D}, \mathbf{a}) \tag{4}$$

$$= \underset{\mathbf{a}\in\mathbf{A}}{\operatorname{argmax}}\ \mathrm{H}(\Phi \mid \mathcal{D}, \mathbf{a}) - \mathbb{E}_{\Theta\sim\Pr(\cdot\mid\mathcal{D})}\left[\mathrm{H}(\Phi \mid \mathbf{a}; \Theta)\right], \tag{5}$$

allowing the computation of entropies to take place in the lower-dimensional label space, Φ. This is known as *Bayesian Active Learning by Disagreement*, or BALD.

The BALD objective invites an appealing interpretation: maximizing the first term encourages selecting an $\mathbf{a}$ that our model is overall uncertain about, and minimizing the second term encourages selecting an $\mathbf{a}$ for which the individual models in $(\theta_1, \ldots, \theta_N)$ can make confident predictions about the outcome ϕ. If we think of the overall uncertainty as a combination of *epistemic* and *aleatoric uncertainty*, then this objective seeks an experiment with high overall uncertainty and low *aleatoric uncertainty*, which therefore has high *epistemic uncertainty*. Intuitively, if the various possible models in $\Pr(\Theta \mid \mathcal{D})$ are individually confident but about differing outcomes, then observing the ϕ value corresponding to $\mathbf{a}$ is likely to prove some of those outcomes incorrect.

Using an ensemble of equally weighted parameter vectors $(\theta_1, \ldots, \theta_N)$ to represent $\Pr(\Theta \mid \mathcal{D})$ allows us to compute a global feasibility prediction,

$$\widehat{\Pr}(\Phi = \phi \mid \mathbf{a}; \Theta) = \frac{1}{N}\sum_{i=1}^{N} \Pr(\Phi = \phi \mid \mathbf{a}; \theta_i)$$

as well as find the experiment that maximizes the estimated BALD objective in the form:

$$\textsc{bald}(\mathbf{a}; \Theta) = \mathrm{H}(\widehat{\Pr}(\Phi \mid \mathbf{a}; \Theta)) - \frac{1}{N}\sum_{i=1}^{N} \mathrm{H}(\Pr(\Phi \mid \mathbf{a}; \theta_i)). \tag{6}$$

C. Sampling Strategies

Now that we have established an informational score for experiments, we consider several sampling strategies for optimizing over $\mathbf{A}$, the set of plans up to a fixed length L. Maximizing the BALD objective over the entire set $\mathbf{A}$ is difficult because we need to consider all discrete plans up to length L, as well as all possible assignments to each continuous abstract action parameter.

Complete One strategy we consider is uniformly sampling complete plans from $\mathbf{A}$ and scoring the samples. We call this the *complete* strategy.

$$\underset{\mathbf{a}\in\mathbf{A}}{\operatorname{argmax}}\ \textsc{bald}(\mathbf{a}; \Theta_{comp}) \tag{7}$$

Unfortunately, to achieve a consistent sampling density, the number of required samples scales exponentially with the length of the plan. Additionally, this strategy might generate most of its samples in the infeasible part of the space.

Greedy Another strategy requiring fewer samples is a *greedy* approach, in which we select the next action a_n which maximizes the BALD objective, given that we have already optimistically constructed $a_{1:n-1}$. This strategy does not take into consideration that if a plan fails early, we do not get to learn from the full plan execution.

We can leverage additional structure afforded to us by the *infeasible subsequence property* when we do active learning using the Θ_{ss} model class. In the following, we consider two possible strategies.

Sequential When generating a potentially informative plan for the Θ_{ss} model, we can take into account the probability a specific action will be attempted (i.e., that the plan was successful up until that action). This allows us to find plans whose informative outcomes have a high probability of being observed. The resulting objective is:

$$\underset{a_{1:n}\in\mathbf{A}}{\operatorname{argmax}} \sum_{i=2}^{n} \Pr(\Phi_{1:i-1} = \mathbf{1} \mid a_{1:i-1}; \Theta_{ss})\textsc{bald}(a_{1:i}; \Theta_{ss}) \tag{8}$$

where the probability is calculated as in Equation 2. This equation computes the expected information gain for executing a plan, taking into account the probability that plan execution fails at any given step as predicted by the learned APF model. We refer to this as the *sequential* strategy.

We implement the *sequential* approach naively by sampling and scoring entire plans. Like *complete*, this method requires exponentially more samples for longer plans, however, in the domain we considered, a sampling approach was acceptable as we had a relatively short plan horizon. In the future work we hope to extend this strategy to a search-based method which prunes candidate plans by their predicted feasibility.

Incremental We also consider a strategy where we only consider plans $(a_1, \ldots, a_n)$ for which we have already observed the prefix to be feasible. In other words, the prefix $(a_1, \ldots, a_{n-1})$ together with result $\phi_{1:n-1} = \mathbf{1}$ are in the current data set $\mathcal{D}$. We call this the *incremental* strategy.

$$\underset{a_{1:n}\in\mathbf{A}}{\operatorname{argmax}}\ \mathbb{1}_{(a_{1:n-1},\phi=1)\in\mathcal{D}}\textsc{bald}(a_{1:n}; \Theta_{ss}) \tag{9}$$

Although this strategy finds more feasible plans than *complete*, it also requires the robot to reuse plan prefixes, so we do not gather novel observations when repeating a plan prefix. Constraining our experiments to build on previously feasible plans may be overly restrictive.

IV. Implementation

We have implemented this framework for a class of problems in which the robot manipulates objects to construct towers. All of our experiments use the 7-DOF Panda robot from Franka Emika, in simulation and in the real world.

A. Domain

The world consists of the robot and a set of objects, $\mathcal{O}$, with which it can interact. In this work, we consider cuboids with non-uniform mass distributions (Figure 3). Each object, $o \in \mathcal{O}$, is described by a tuple, (d, c, m), where $d \in \mathbb{R}^3$ are the dimensions, $c \in \mathbb{R}^3$ is the offset of the center of mass from the center of geometry, and $m \in \mathbb{R}$ is the object's mass.

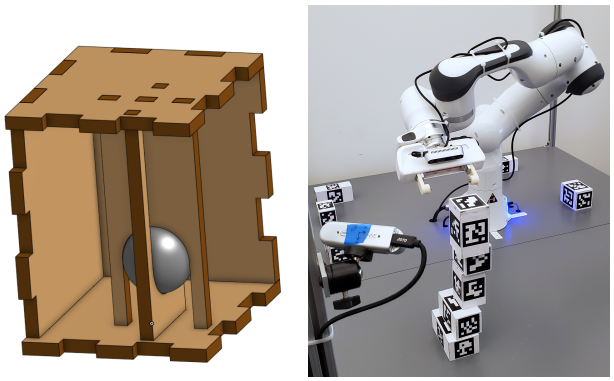


Fig. 3: Left: The cuboids for this manipulation task were constructed from laser cut plywood. A 25mm diameter lead ball is mounted inside some of the objects to significantly alter the mass distribution. Right: Unique ArUco markers are applied to each face of each object, for object identification and localization. Visible in the foreground is one of the two external cameras mounted around the workspace.

During the experimentation phase we use a set of 10 blocks, and all evaluations are performed with a different set of 10 blocks. The block parameters from each set are sampled from the same uniform distribution over the dimensions of the objects and the locations of the center of mass within the objects.

The abstract actions are to place objects onto a stack; they are specified by $a = (o, r)$ where $r \in SE(3)$ is the relative pose of o with respect to the object placed in the previous action (or the table if this is the first object placement).

An abstract plan is feasible if the detailed planning and execution system can find and execute robot commands (i.e., grasp poses and motion plans) such that the objects are placed on top of one another and the resulting tower is stable. Note that, for this property to hold, each prefix of the plan must also be feasible — that is, each subtower is stable.

The learned APF model is applied to three different objectives in the *evaluation phase*:

1) *Tallest Tower:* The objective is to construct the tallest possible tower.
2) *Longest Overhang:* The objective is to construct the tower with the maximum distance from the center of geometry of the bottom block to the furthest vertical side of the top block.
3) *Maximum Unsupported Area:* The objective is to construct the tower where each block has as much area possible unsupported by the block below it.

See Appendix C for a discussion on the generalizability of our method outside of the towers domain.

B. Perception, planning, and execution

Perception For the system to robustly pick up blocks and recover from unstable towers falling in unpredictable configurations, we require a perception system that can identify and localize objects at arbitrary positions. Although more advanced perception systems might be needed for arbitrary objects, vision is not the immediate focus of this work, so we pattern our objects with ArUco markers to simplify perception. To indicate identity and avoid orientation ambiguity, each object has a unique ArUco marker on each face.

Two RealSense D435 depth cameras mounted statically on a frame observe the workspace and allow for localizing the objects with minimal occlusions. If an object is not visible, it is assumed to be at its home position behind the arm. Due to the resolution of the cameras and size constraints of the tags on the blocks, we found that the pose estimates from the static cameras can have up to 1 centimeter of error. This level of error is acceptable, as rough pose estimates are refined with a third RealSense D435 camera mounted on the robot wrist. As the arm moves to a pre-grasp pose computed from the noisy object pose estimate, the wrist-mounted camera collects images closer to the the object to be grasped, allowing for a refined pose estimate and more precise grasp.

Planning In this domain, executing an abstract action requires a multi-step task and motion plan. For example, to place an object on a tower with an arbitrary relative pose to the block below it, regrasping may be necessary. When a tower falls over, the robot will also need to move fallen blocks out of the way to build the next tower.

To handle these scenarios, we turn to a large body of work in task and motion planning. We use PDDLStream [13], which integrates PDDL task-level planning with lower level motion planning. The PDDL domain describes actions including picking and placing objects and moving the arm through free space, while a Bidirectional RRT [23] in joint configuration space performs motion planning and collision checking with a surrogate world model implemented in PyBullet [7]. In this work we make the simplifying assumption that there are dedicated positions for the base of the tower, regrasping, and storage for each of the objects. These constraints are specified to the planner to reduce planning time by limiting the search space of possible action parameters.

Execution The motion plans generated by PDDLStream are executed using joint-space controllers on the robot. When constructing a tower in the experimentation phase, after each block placement the wrist camera is used to check for tower stability. If a tower was unstable, then the last manipulated block will not be near its expected pose in front of the gripper.

To improve data collection efficiency, we parallelize execution and planning. As the robot executes a motion plan to assemble or disassemble a tower, the planner produces plans to move each individual block in that tower under the assumption that all actions will be successful. This parallelism is interrupted if a tower falls over prematurely, prompting a replan to clear the fallen blocks.

If the state of the world is such that a robot is unable to find a plan to proceed with experimentation or execute an existing plan, human intervention may be required. We have provisioned for several of these cases, including blocks falling off the table, or too close to one another for the planner to find feasible grasp candidates. Once such issues are manually resolved (e.g., by putting the block back on the table), the robot can update its estimate of block poses and resume planning.

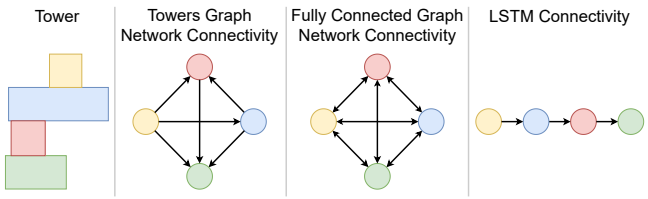


Fig. 4: The flow of information for the model architectures compared in Section V-B. The TGN uses domain specific connectivity, while the FCGN assumes no prior knowledge on how the blocks should be used together to inform predicting feasibility. The LSTM simply iterates through the blocks starting with the top block.

C. Learning

As discussed in Section III-A, the distribution over APF model parameters is represented by an ensemble of networks. In our implementation, the ensemble is made up of GNNs with domain-specific connectivity. The input to the GNN is an abstract action sequence, $\mathbf{a}$. Each action, or block placement a_i, is passed into a separate node in the graph, and each node (corresponding to a block placement in the tower) is connected to nodes above it in the tower. This mirrors the analytical computation of tower stability, in that each subtower's combined center of mass must be within the contact patch of the block below it. We refer to this network architecture as a TGN, or *Towers Graph Network*. Other network architectures which are invariant to task plan length are a *Fully Connected Graph Network* (FCGN) and a LSTM model. The FCGN uses the same node and edge networks as our TGN model, but has edges between each node. The LSTM model passes each block's vector representation through the network in order starting from the top of the tower, and most closely matches our TGN connectivity. A visualization of the connectivity of each architecture is given in Figure 4.

In our experiments, 10 networks are used in the ensemble. Each individual network is randomly initialized and trained using the binary cross-entropy loss function with early stopping according to the loss on a validation set, which is also collected actively.

Before active experimentation, each model in the ensemble is initialized by training on the same dataset (shuffled differently for each) of 40 randomly generated towers. For the *complete* strategy the towers are of size $2-5$, and for the *sequential*, *incremental*, and *greedy* strategies they are of size 2. During the experimentation phase, at each iteration the top 10 most informative towers are chosen and labeled by attempting to build each with the robot. 20% of the collected data is added to a validation set and the remainder is added to the training set. We perform data augmentation by rotating each collected tower $90, 180, \text{and}, 270$ degrees about the axis normal to the table surface.

V. Evaluation

In the following sections we evaluate the utility of various sampling strategies (Section V-A) and model architectures (Section V-B) for increasing data efficiency and generalization. In simulation, we show that these choices greatly influence the robot's accuracy and performance on multiple downstream tasks. In addition, Section V-C gives the performance of our *incremental* sampling strategy on a real robot. We show that not only can the robot learn an APF model from real data and use it to perform downstream tasks, but also that learning on the real robot allows us to be robust to noise that would be difficult to model in simulation.

When evaluating task performance, we randomly select 5 blocks from a set of novel blocks and execute the best tower (that uses all 5 blocks) found by the Monte Carlo planner, given the task objective and our *Learned* APF model. The reward received from executing this tower is used to calculate normalized regret, which is the difference between this received reward and the largest reward of a stable tower considered by the planner (found using an *Analytical* model). If a tower is unstable, we assign a reward of zero.

A. Impact of Sampling Strategy

Figure 5 shows the task performance of models trained using the four sampling strategies described in Section III-C. Each strategy has results aggregated from 4 independent training runs and 50 task evaluations per run all performed in a simulated environment. Different sets of blocks are used between training and evaluation.

Our APF model performs best on the *Tallest Tower* task, successfully minimizing regret after constructing only 200 towers with the *sequential* method. The *incremental* method also performs well, successfully minimizing regret with minimal variance across runs. The *complete* method on average performs well, but has very high variance for the more challenging tasks, *Longest Overhang* and *Maximum Unsupported Area*. Note that the shaded region represents the quartile distribution — a region that extends to 1.0 means that more than a quarter of the trials were unstable. This highlights the importance of considering the *infeasible sub-sequence property* when sampling and scoring plans in the action space. Finally, the naive *greedy* strategy performs the worst, and is only able to achieve decent performance on the *Tallest Tower* task after seeing roughly 800 training towers, likely due to the fact that it is not considering the feasibility of subtowers when searching the actions space, just greedy single-block placements.

The *Maximum Unsupported Area* and *Longest Overhang* tasks are more challenging for the robot because they require deep understanding of the tower stability decision boundary, while the *Tallest Tower* task only requires a rough understanding of how to build stable towers with high confidence. These results show that in spite of the difficulty of the first two tasks, the active learner is able to improve its understanding of the decision boundary well enough to perform tasks with very low regret and low variance.

In Appendix B we give additional results which compare against baselines that do not leverage active learning.

B. Model Architecture and Generalization

We compare our TGN to the other network architectures discussed in Section IV-C and shown in Figure 4. For this

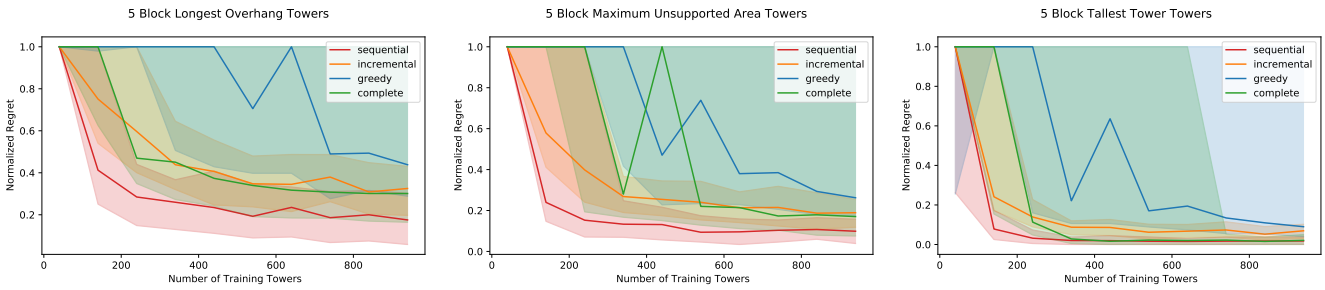


Fig. 5: A comparison of sampling strategies on different downstream tasks all performed in simulation. Each method evaluation consists of 4 separate APF model-learning runs, and each point is the Median Normalized Regret of 50 individual planning runs per learned model. The shaded regions show 25% and 75% quantiles.

evaluation, we report accuracy on a held-out test set of towers built with a novel set of blocks. The test set consists of half feasible and half infeasible towers, and 1000 towers for each tower size. Our models were trained in simulation on towers consisting of up to 5 blocks, but our results give model accuracy for towers ranging from 2 to 7 blocks, shown in Figure 6.

In the towers domain, it is necessary to consider the joint centers of mass for groups of blocks above support blocks. While the LSTM architecture could remember the previous blocks as it iterates through the tower, in practice we find that it is outperformed by the graph network architectures. We believe this is because the connectivity of the graph networks allows them to precisely compare adjacent blocks in addition to aggregating information about multiple blocks. The weakness of the LSTM is more pronounced as the number of blocks in a tower increases.

Our TGN architecture is structured to be biased towards our particular domain, so it is able to improve its predictions much faster than the other architectures. This enables good planning time performance as seen in Section V-A with similar long-term performance to the FCGN architecture.

C. Real Robot Experiments

Finally, we give results for executing the entire active learning pipeline on a real Panda robot (see Section IV for details of the real robot setup). In total, the robot built 400 towers while training over a period of 55 hours.

For the experimentation phase, we used a fixed set of 10 training blocks. The TGN ensemble is initialized with 40 random 2-block towers labeled in simulation with added relative-pose noise. We generated candidate experiments using the *incremental* strategy described in Section III-C, and produced stability labels for the constructed towers by observing the outcome with the cameras.

During the evaluation phase, we test the robot's ability to use its learned APF model to perform all tasks described in Section IV-A with a separate set of 10 held-out evaluation blocks. We compare the learned APF model to two baselines. First, we compare to a hand-engineered model of plan feasibility that calculates whether a candidate tower is feasible in a noiseless world. We also compare to a noisy simulator feasibility model, which predicts a tower is feasible only if the candidate tower is also stable to 10 normally distributed perturbations, with a standard deviation of 5mm, for each block placement.

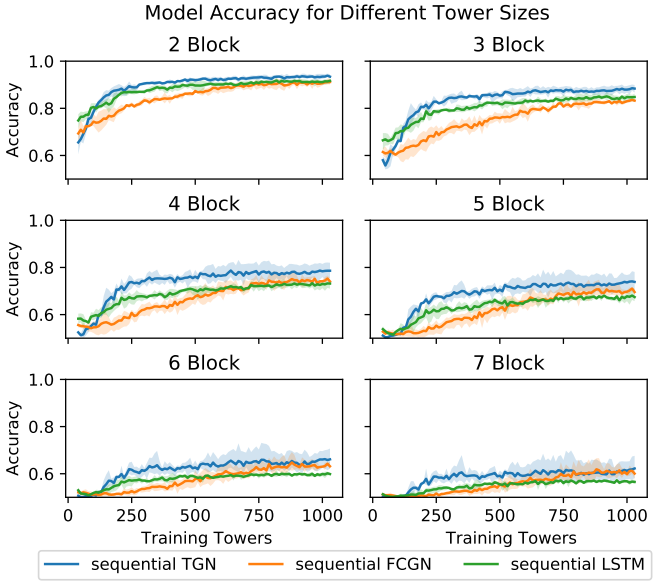


Fig. 6: A comparison of different network architectures using the *sequential* sampling strategy, performed in simulation. The accuracy for each method is averaged over 3 separate training runs. The shading shows the minimum and maximum accuracy from these runs.

For each task and model we select 5 blocks at random from the evaluation set and plan to maximize the given objective. If a constructed tower is unstable, it gets zero reward. The robot constructs 10 towers for each task and feasibility model. In Table I, we report average normalized regret, and the number of total trials that resulted in a stable tower. Figure 7 includes

	Tallest Tower		Longest Overhang		Max Unsupported Area	
Model	**Regret**	**#Stable**	**Regret**	**#Stable**	**Regret**	**#Stable**
Analytical	0.30	7/10	0.80	2/10	0.80	2/10
Simulation	0.41	6/10	0.47	8/10	0.19	10/10
Learned	0.15	9/10	0.45	9/10	0.33	9/10

TABLE I: Real robot task performance when using different APF models. The *Learned* model was trained with data collected through active learning on the real Panda robot. The *Analytical* and *Simulation* models calculate feasibility using the known underlying dynamics but use either no noise or simple noise models respectively.

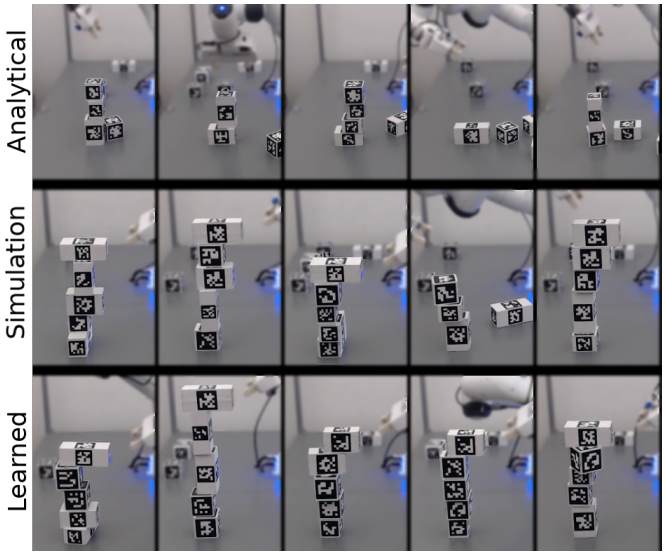


Fig. 7: Towers built for the *Longest Overhang* task when using different APF models. Observe that some towers built using the *Analytical* and *Simulation* models were unstable. See the appendix for more examples.

images of the robot performing the *Longest Overhang* task using the different models for APF.

From these results, it can be seen that the *Analytical* model can build towers with high reward when the tower is stable, but the towers it chooses to build are rarely stable across all three tasks. However, the *Simulation* and *Learned* APF models can still build towers with large overhang while considering the effects of noisy action execution on a real robot. Our *Learned* model performs competitively with the *Simulation* model, and in aggregate leads to similar stability across all tasks (27 versus 24 stable towers out of 30). However, note that the *Simulation* model presents higher variability in tower stability across tasks. This is because the model makes assumptions about the type of noise distribution (Gaussian only in the plane normal to the table) and its parameters (mean and variance), which may be more suitable for certain tasks. The *Learned* model, on the other hand, may capture other complex real-world phenomena that significantly contribute to plan feasibility in a task-agnostic setting.

VI. Related Work

Hierarchical Planning It is a common approach to long-horizon planning problems; decomposing the solution into high level reasoning over *abstract actions* and lower level reasoning over *concrete actions* [25, 31, 34]. Singh and Kelly [33] performed early work on considering the feasibility of high level plans to improve efficiency when planning in high dimensional spaces. Recent works have proposed methods that predict feasibility of an action as a way to reduce the number of calls to expensive solvers and enable more efficient planning [8, 36]. Our work builds upon this literature by presenting a method to learn feasibility actively when detailed physical models are not available.

Learning for Task and Motion Planning In this work we leverage a task and motion planning framework to plan for concrete actions, then execute those actions to determine plan feasibility. Others have explored learning the feasibility of actions, specifically an action's preconditions and effects. [35] uses a Gaussian process to learn these action parameters, with a specialized acquisition function to guide learning. Our work differs in that our acquisition strategy considers observations of entire plans, which means our active learning is non-myopic. In addition, for the stacking domain we found graph neural networks to be a more applicable class of function approximators in which plan length can vary. A similar approach is taken in [21], a precursor to [35].

Active Learning Active learning [26] is a well-established learning paradigm that aims to minimize the number of samples needed to learn the target concept. Recently, Gal et al. [12] have extended BALD [19] to complex model domains of deep neural networks using MC-dropout [11]. However, in this work, we follow the approach of Beluch et al. [3] and use an ensemble of deep networks for active learning.

Ideas from active learning have been used for efficient learning in model-based reinforcement learning tasks [28, 32]. Pathak et al. [28] explore the environment by taking actions that maximize disagreement between an ensemble of forward models. Instead of predicting continuous states, our work learns a model that predicts feasibility of *abstract plans* where the focus is on performing long-horizon tasks.

Learning Dynamics Many recent works have focused on learning predictive dynamics models in scenes with varying numbers of objects. In such scenarios, it has been shown that explicitly representing objects and their relations in the model can lead to more efficient learning and generalization [1, 2, 5, 38]. As such, graph networks are becoming a more common modeling choice in these domains [22, 30]. Specifically, in a stacking domain, Hamrick et al. [18] show that a graph neural network can predict stability properties of towers when given access to a large training set.

Stacking Domain Tasks that involve stacking objects in a *Blocks World* have a long history in artificial intelligence. Early works developed methods to compute the stability of block placements and construct a target configuration [4, 37]. More recently, computer vision researchers have developed scene understanding algorithms that take into account known geometries and stability properties of objects within the scene [15, 20, 29]. Furrer et al. [10] developed a system that can build stacks out of stones using detailed models of the objects.

Recent work has shown the ability to predict tower stability using deep learning techniques [14, 18, 24]. However, typically these works have used passively collected datasets which include orders of magnitude more samples than required in this work — making them infeasible to actively collect on a real robot. An active learning approach allows the robot to explore efficiently, and learn a feasibility model under the real-world noise distribution. In addition, vision-based systems would not be effective when the state contains non-visual properties, such as the center of mass in our stacking domain [14, 24].

VII. Conclusion

We have presented a system which leverages information-theoretic active learning to acquire an *Abstract Plan Feasibility* model, and shown that incorporating plan feasibility into the active learning strategy can dramatically improve sample efficiency. We deployed our system on a real Franka Emika Panda robot arm in a block-stacking domain, enabling the robot to learn a useful APF model with only 400 experiments.

In future work, we are eager to apply this approach to other domains. We believe that this self-supervised method of curious exploration is an exciting direction, as it may someday allow the millions of robots sitting powered-off in laboratories around the world to make effective use of their downtime.

Acknowledgments

The authors would like to thank Rachel Holladay and Caelan Garrett for contributing their time to help us set up our Panda robot arm and get our footing using the Franka Emika and PDDLStream software. This research was generously sponsored by Honda Research Institute.

References

[1] P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences (PNAS)*, 110(45):18327–18332, 2013.

[2] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[3] W. H. Beluch, T. Genewein, A. Nurnberger, and J. M. Kohler. The Power of Ensembles for Active Learning in Image Classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] M. Blum, A. Griffith, and B. Neumann. A stability test for configurations of blocks. 1970.

[5] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research (JAIR)*, 4:129–145, 1996.

[7] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[8] D. Driess, O. Oguz, J. S. Ha, and M. Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[9] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. *An analysis of approximations for maximizing submodular set functions—II*, pages 73–87. Springer Berlin Heidelberg, 1978.

[10] F. Furrer, M. Wermelinger, H. Yoshida, F. Gramazio, M. Kohler, R. Siegwart, and M. Hutter. Autonomous robotic stone stacking with online next best object target pose planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[11] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*. PMLR, 2016.

[12] Y. Gal, R. Islam, and Z. Ghahramani. Deep Bayesian Active Learning with Image Data. In *International Conference of Machine Learning (ICML)*, 2017.

[13] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. PDDLStream: Integrating symbolic planners and black-box samplers via optimistic adaptive planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2020.

[14] O. Groth, F. B. Fuchs, I. Posner, and A. Vedaldi. Shapestacks: Learning vision-based physical intuition for generalised object stacking. In *European Conference on Computer Vision (ECCV)*, 2018.

[15] A. Gupta, A. A. Efros, and M. Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *European Conference on Computer Vision (ECCV)*, 2010.

[16] J. S. Ha, D. Driess, and M. Toussaint. A probabilistic framework for constrained manipulations and task and motion planning under uncertainty. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[17] N Haghtalab, S Mackenzie, AD Procaccia, O Salzman, and S Srinivasa. The provable virtue of laziness in motion planning. *International Conference on Automated Planning and Scheduling (ICAPS)*, 2018.

[18] J. B. Hamrick, K. R. Allen, V. Bapst, T. Zhu, K. R. McKee, J. B. Tenenbaum, and P. W. Battaglia. Relational inductive bias for physical construction in humans and machines. In *the Annual Meeting of the Cognitive Science Society (CogSci)*, 2018.

[19] N. Houlsby, F. Huszar, Z. Ghahramani, and M. Lengyel. Bayesian Active Learning for Classification and Preference Learning. In *NeurIPS Workshop on Bayesian optimization, experimental design and bandits: Theory and applications*, 2011.

[20] Z. Jia, A. C. Gallagher, A. Saxena, and T. Chen. 3D Reasoning from Blocks to Stability. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2015.

[21] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Learning composable models of parameterized skills. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 886–893. IEEE, 2017.

[22] B. Kim and L. Shimanuki. Learning value functions with relational state representations for guiding task-and-motion planning. In *Conference on Robot Learning (CORL)*. PMLR, 2020.

[23] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[24] A. Lerer, S. Gross, and R. Fergus. Learning physical intuition of block towers by example. In *International Conference on Machine Learning (ICML)*, pages 430–438. PMLR, 2016.

[25] T. Lozano-Pérez and L. P. Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2014.

[26] D. J. C. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):590–604, 1992.

[27] P. Oudeyer, F. Kaplan, and A. Hafner, V.and Whyte. The playground experiment: Task-independent development of a curious robot. In *Proceedings of the AAAI Spring Symposium on Developmental Robotics*, 2005.

[28] D. Pathak, D. Gandhi, and A. Gupta. Self-supervised exploration via disagreement. In *International Conference on Machine Learning (ICML)*, pages 5062–5071. PMLR, 2019.

[29] L. G. Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

[30] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning (ICML)*. PMLR, 2018.

[31] Y. Shoukry, P. Nuzzo, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada. Scalable lazy smt-based motion planning. In *IEEE Conference on Decision and Control (CDC)*, 2016.

[32] P. Shyam, W. Jaśkowski, and F. Gomez. Model-based active exploration. In *International Conference on Machine Learning (ICML)*, pages 5779–5788. PMLR, 2019.

[33] S. Singh and A. Kelly. Robot planning in the space of feasible actions: two examples. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, 1996.

[34] M. Toussaint and M. Lopes. Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[35] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Active model learning and diverse action sampling for task and motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4107–4114. IEEE, 2018.

[36] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki. Learning feasibility for task and motion planning in tabletop environments. *IEEE Robotics and Automation Letters (RA-L)*, 4(2), 2019.

[37] P. Winston. The mit robot. In *Machine Intelligence*, volume 7, 1972.

[38] V. Xia, Z. Wang, K. Allen, T. Silver, and L. P. Kaelbling. Learning sparse relational transition models. In *International Conference on Learning Representations (ICLR)*, 2018.

[39] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

APPENDIX

A. Real Robot Evaluations

In this section we provide additional results for the experiments described in Section V-C. Figure 8 shows the towers built when performing the *Tallest Tower* task with each of the analytical, simulation, and learned models. 10 towers were built for each task using 5 blocks. Towers that have fallen over or do not have 5 blocks were infeasible plans. Figures 9 and 10 show the evaluation towers for the *Longest Overhang* and *Maximum Unsupported Area* tasks, respectively.

B. Supervised vs Active Learning Comparisons

In this section we compare the active learning methods described in this paper to supervised learning baselines which do not perform active data collection to train the APF model. Figure 11 compares our methods which use the Θ_{ss} model class to a *random-ss* strategy. *random-ss* randomly samples action sequences and labels each subsequence. The results show that our active methods which leverage the *infeasible subsequence property*, *sequential* and *incremental*, outperform *random-ss*. *random-ss* only outperforms *greedy*, showing that a random strategy is able to find more interesting training towers than a myopic approach which simply tries to maximize the BALD objective for a single block placement.

Figure 12 compares our method, *complete*, which uses the Θ_{comp} model class, to a *random-comp* strategy. *random-comp* randomly samples action sequences to train on and only labels the full sequences. As expected, training on sequences which maximize the BALD objective is more effective than randomly sampling action sequences.

The data used to train the *random* methods is not actively collected, but we show how training on increasingly larger datasets compares to the actively collected dataset.

C. Method Generalizability

Here, we motivate a more general class of problems for which our system applies and clarify which components of the system are specific to our chosen domain. Our method most benefits domains where the feasibility of an action depends strongly on the preceding action sequence (e.g., adding a fifth block to a tower that already has four blocks). Domains that include construction tasks, like ours, will commonly benefit from non-myopic information gathering and a feasibility predictor that incorporates previous actions into its predictions. To apply our method to a new domain (e.g., consider a packing problem where many objects need to be placed in a larger container), the overall system/methodology would remain unchanged. However, one would need to adapt the following domain-specific components:

1) *Abstract action definitions*. Parameterize an abstract action that is appropriate for the domain and connect it to concrete actions (e.g., object locations within the container).
2) *Experimental infrastructure*. Additional capabilities to autonomously plan and execute abstract actions in the physical world (e.g., motion and grasp planning infrastructure).
3) *Feasibility detector*. The notion of APF will depend on the desired outcome of an abstract action. For a new action, we require a method to autonomously acquire the feasibility label during execution (e.g., whether the gripper will collide when placing the object or if an object will be damaged).
4) *Additional inductive bias* (optional). Additional structure to the learner can further increase data efficiency, as shown by our TGN method. However, this is not required as we show in Figure 4 that a general purpose graph network can be used (e.g., a graph network that has connectivity between all objects).

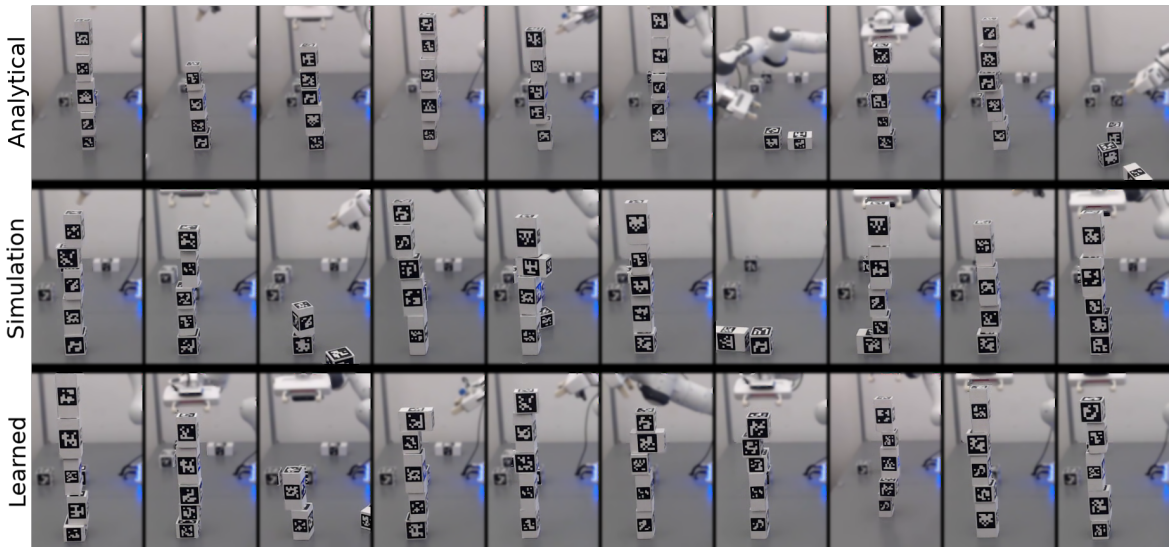


Fig. 8: Towers built for the *Tallest* task when using different *Abstract Plan Feasibility* models.

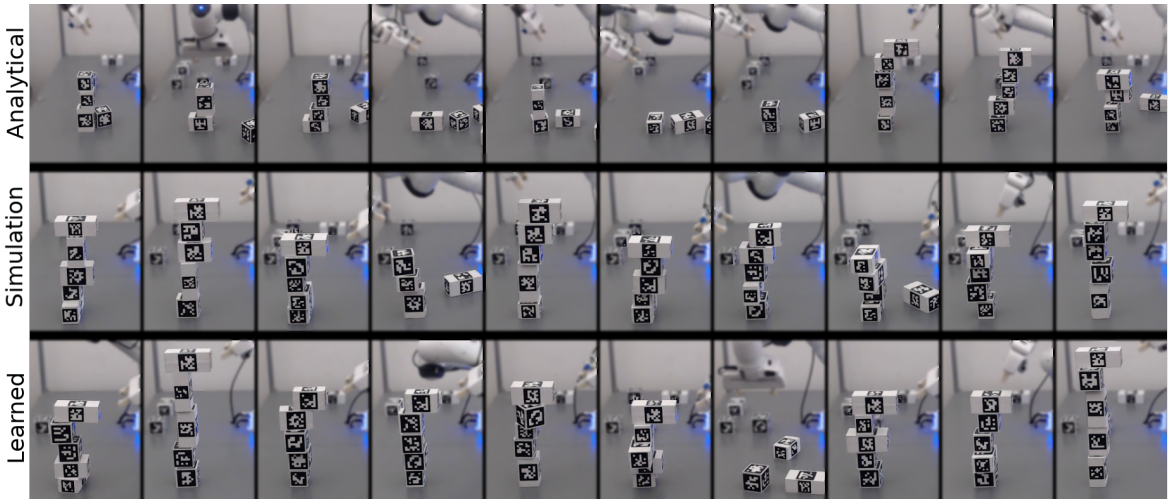


Fig. 9: Towers built for the *Longest Overhang* task when using different *Abstract Plan Feasibility* models.

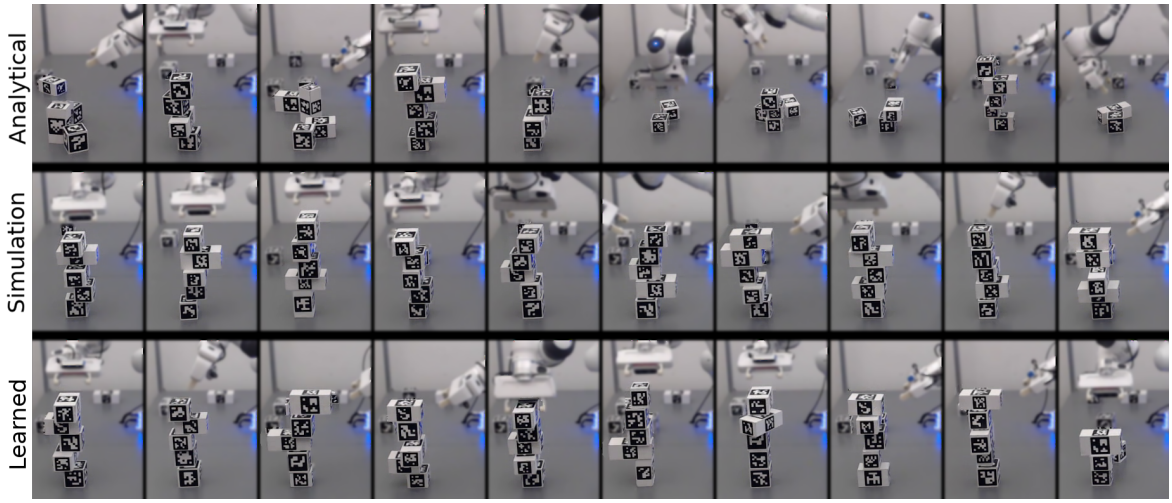
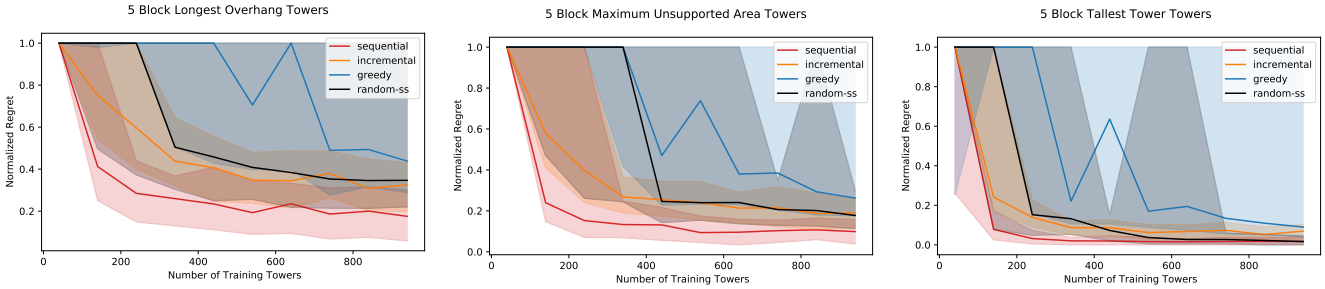


Fig. 10: Towers built for the *Maximum Unsupported Area* task when using different *Abstract Plan Feasibility* models.



(a)

Fig. 11: A comparison of sampling strategies which use the Θ_{ss} model class on different downstream tasks all performed in simulation. The *random-ss* strategy randomly samples action sequences and thus does not use active learning. Each method evaluation consists of 4 separate APF model-learning runs, and each point is the Median Normalized Regret of 50 individual planning runs per learned model. The shaded regions show 25% and 75% quantiles.

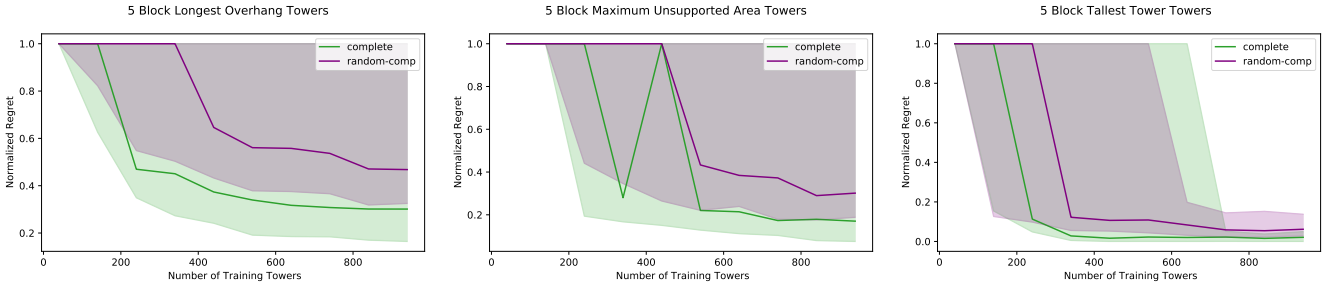


Fig. 12: A comparison of sampling strategies which use the Θ_{comp} model class on different downstream tasks all performed in simulation. The *random-comp* strategy randomly samples action sequences and thus does not use active learning. Each method evaluation consists of 4 separate APF model-learning runs, and each point is the Median Normalized Regret of 50 individual planning runs per learned model. The shaded regions show 25% and 75% quantiles.