Sequence-Based Plan Feasibility Prediction for Efficient Task and Motion Planning

Zhutian Yang
MIT
ztyang@mit.edu

Caelan Reed Garrett
NVIDIA
cgarrett@nvidia.com

Dieter Fox
NVIDIA
dieterf@nvidia.com

Abstract:
We present a learning-enabled robot Task and Motion Planning (TAMP) algorithm that generates diverse plan skeletons and sorts them by their feasibility, i.e., the likelihood of finding values for the action parameters that satisfy all geometric constraints. We propose PIGINet, a novel Transformer-based architecture that predicts plan feasibility by tokenizing the plan skeleton, goal condition, and initial state as a sequence, fusing image, text, and value embeddings. We evaluate the runtime of our learning-enabled TAMP algorithm on several distributions of kitchen rearrangement problems, comparing its performance to that of non-learning baselines. Our experiments show that PIGINet substantially improves planning efficiency, cutting down runtime by 80% on average on pick-and-place problems with articulated obstacles. It also achieves zero-shot generalization to problems with unseen object categories thanks to its visual encoding of objects.

Keywords: Task and motion planning, learning-to-plan, mobile manipulation

1 INTRODUCTION

Robots planning long-horizon behavior in complex environments must be able to quickly reason about the impact of the environment's geometry on what plans are feasible. Many Task and Motion Planning (TAMP) [1] algorithms accomplish this by balancing the computational time spent on two processes: 1) finding high-level plan skeletons that satisfy symbolic conditions; and 2) generating continuous action parameter values that satisfy geometric constraints through sampling or optimization. This balancing act is particularly challenging when the robot configuration space is disconnected due to obstruction caused by manipulable obstacles. For example, a mobile robot may be asked to rearrange food items among fridges, cabinets, and tables. Multiple doors and other food items may be blocking the paths to reach the goal objects. In these problems, the number of infeasible candidate plan skeletons increases exponentially as the planning horizon and number of objects increases. An uninformed TAMP algorithm would waste a substantial amount of time attempting to satisfy many unsatisfiable plan skeletons before working on the feasible ones.

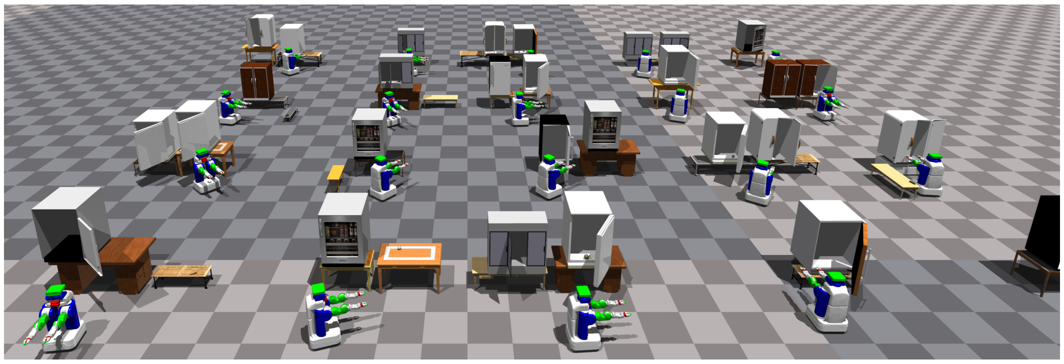


Figure 1: Example rearrangement problems in our dataset, rendered in IsaacGym. Here, articulated objects, such as doors, prevent the robot from directly picking and placing objects.

In this work, we present a learning-enabled TAMP algorithm that predicts if a plan skeleton is feasible before attempting to refine it. Our model, ***PIGINet***, uses a Transformer architecture to encode 1) a candidate ***P***lan skeleton with ***I***mage features of objects, 2) the ***G***oal formula, and 3) relations and continuous values in the ***I***nitial state. PIGINet outputs a probability that the plan skeleton is feasible. The elements of each action or relation, such as their names, objects, poses, and joint angles, are processed to produce embeddings of the same length and fused together through averaging. We deploy our learner in a TAMP algorithm that sorts plan skeletons by the learner's predicted likelihoods of feasibility. We evaluate the runtime of our learning-enabled TAMP algorithm on several distributions of unseen problem instances in comparison to a baseline and ablations. Our experiments show that learning to predict skeleton feasibility can substantially improve planning performance, cutting down runtime by 80% on average on pick-and-place problems with articulated obstacles.

2 RELATED WORK

Task and Motion Planning (TAMP) Two existing TAMP algorithms use *diverse planning* techniques [2], which identify multiple distinct plans for a given problem, to produce candidate plan skeletons to be refined using sampling [3] or optimization [4]. Because diverse planners are unaware of the semantics of predicates, many candidate plans have the same sources of infeasibility. As a result, these TAMP algorithms waste time finding continuous values for similar, unsatisfiable plan skeletons. Many TAMP algorithms contain mechanisms that provide specific feedback [1] to the search over plan skeletons using failed motion queries [5] or unsatisfiable constraint sets [6]. However, all of these approaches require expensive geometric planning to first identify failed skeletons and second generate the feedback.

Learning to speed up TAMP Silver *et al.* [7] developed a Graph Neural Network (GNN) approach that ignores irrelevant objects, taking into account object obstruction. Khodeir *et al.* [8] extended this approach to predict which optimistic parameter values constructed through lazy sampling should be considered. Because these approaches condition only on the initial and goal state instead of candidate plans, the learners carry the burden of generalizing over problems that vary substantially in solutions, which is often challenging. By instead providing a learner candidate plans, our approach is able to more easily generalize over this distribution of problems.

Several learning-for-TAMP approaches learn single-action feasibility classifiers using tabular [9], depth image [10, 11, 12], or point cloud [13] encodings of the environment. Kim *et al.* [14] learned a cost-to-go heuristic estimator using a relational embedding of the state. Because they consider individual actions on one plan at a time, any free parameters present in the candidate actions, such as robot configurations, object placements, and object grasps, must already be assigned via prediscretization or sampling. In contrast, our approach is able to consider full plans all at once, allowing it to account for parameters that persist over time in other states and actions.

Multi-modal inputs for robotic manipulation We are inspired by recent works used attention-based neural networks to encode the state, fusing multi-modal inputs, and make object-centric decisions. Zhu *et al.* [15] encode states as symbolic and geometric scene graphs and then process them with GNNs separately to generate the next action name and action parameters. Liu *et al.* [16] encode text and visual embeddings in the same sequence for a Transformer encoder by adding each embedding with a corresponding type and positional encoding. Blukis *et al.* [17] also uses a Transformer encoder to process a sequence of subgoals consisting of action types and arguments into a vector embedding and combines them with a language embedding through a dense MLP for subgoal generation. Yuan *et al.* [18] learn object embeddings by encoding a sequence consisting of image patches of the whole scene and canonical views of each objects in a Transformer encoder-decoder architecture and then use the learned embedding for querying object spatial relations or a direction for gripper movement.

3 PROBLEM FORMULATION

We define a TAMP *domain* $\langle \mathcal{P}, \mathcal{A} \rangle$ by a set of *predicates* $\mathcal{P}$ and a set of *actions* $\mathcal{A}$, both of which can be viewed as tuples with a name, discrete arguments such as objects, and continuous arguments such as poses (*e.g.* p_0), grasps (*e.g.* g_1), robot configurations (*e.g.* q_0), joint angles (*e.g.* a_0), and trajectories (*e.g.* t_1). A TAMP *problem* $\langle \mathcal{O}, \mathcal{I}, \mathcal{G}, \mathcal{A} \rangle$ is defined by a set of manipulable *objects* $\mathcal{O}$, an *initial state* $\mathcal{I}$, and a conjunctive set of *goal literals* $\mathcal{G}$. Table 1 shows some examples of each consturct. Note that fridge1:door1, fridge1:space1, and fridge1 are three different objects in the planning problem, since they afford different actions.

$\mathcal{O}bj$	$\mathcal{I}nit$	$\mathcal{G}oal$
tomato1	`Graspable`(tomato1)	`Holding`(tomato1)
fridge1:door1	`IsJoint`(fridge1:door1, fridge1)	`Closed`(fridge1:door1)
fridge1:space1	`Containable`(tomato1, fridge1:space1)	`In`(tomato1, fridge1:space1)
table2	`Supported`(tomato1, table2, p_0)	`On`(tomato1, table2)

Table 1: Example objects ($\mathcal{O}$), initial facts ($\mathcal{I}$), goal conditions ($\mathcal{G}$) and `predicates` ($\mathcal{P}$)
.

A *solution* is a finite sequence of action instances that, when sequentially applied to the initial state $\mathcal{I}$, produces a terminal state where the goal literals $\mathcal{G}$ all hold. When the continuous arguments of a solution are unbound (denoted by the prefix *?*), we call it a $\mathcal{S}keleton\ \pi$. *Refinement* is the process of finding continuous values for unbound parameters in a skeleton that satisfy the constraints among those values, such as inverse kinematics and collision-free constraints. A skeleton can be infeasible for a variety of reasons, for example, the object is not reachable from any grasp, or there is no collision-free arm trajectory that ends with the gripper at grasp pose. To predict how likely it is to refine a skeleton, we strip away the continuous arguments to make an action tuple as input to the learning model that includes only object arguments, as shown on the right of Table 2.

$\mathcal{S}keleton\ \pi$	π for PIGINet
`move`(q_0, $?q_1$)	(`move`)
`pullopen`(fridge1:door1; a_0, $?a_1$, $?g_1$, $?t_1$)	(`pullopen`, fridge1:door1)
`move`($?q_1$, $?q_2$)	(`move`)
`pick`(tomato1; p_0, $?q_2$, $?g_2$, $?t_2$)	(`pick`, tomato1)
`move`($?q_2$, $?q_3$)	(`move`)
`place`(tomato1; $?p_1$, $?q_3$, $?g_2$, *?t4*)	(`place`, tomato1)

Table 2: An example $\mathcal{S}keleton\ \pi$, and simplified action tuples used for plan feasibility prediction.

The role of the plan feasibility predictor f is to take in an initial state $\mathcal{I}$, a skeleton π, and goal conditions $\mathcal{G}$ and then predict the likelihood that there exists a solution to π. We use f as a scoring function for ranking a batch of skeletons for refinement:

$$f(\mathcal{I}, \pi, \mathcal{G}) \rightarrow [0, 1].$$

4 LEARNING METHOD

Given a problem and a plan, PIGINet first builds a dictionary of embeddings for objects, continuous values, and text (predicate and action names). Next, it encodes each literal in the initial state and the goal, as well as each action in the plan, as a token in the input sequence for a Transformer encoder. The decoded output is the probability of plan feasibility. The architecture is shown in Figure 2.

For objects, we segment the rgb image of the scene to get image crops of individual objects and salient large parts, process them with a pre-trained vision network [19], then reduce the dimension of the extracted features with a three-layer Multi-Layer Perceptron (MLP). All objects are rendered with solid colors, and segmentation is perfect in simulation. Doors and fridge bodies are different segments. If the objects are occluded, none of its pixels will show except for the background color. **For continuous variables** in the initial state, we directly transcribe the values of object poses, object joint angles, and the robot base configurations. We concatenate the one-hot encoding of the variable type with the value of the variable. We zero-pad the value tuples such that digits of different value types occupy different slots. We process the resulting feature vector with a linear and a ReLU layer. **For text**, which are rephrases of the names of predicates or actions in the domain, we encode them with a pre-trained language network [19], then fine-tune the features through a linear and a ReLU layer. Rephrasing helps deal with strange names such as `isjointto` and `pull-open`, which are rewritten to "*this is a joint of that object*" and "*pull the handle of door joint*".

For each literal in $\mathcal{I}nit$ and $\mathcal{G}oal$, as well as each action in π, we add up the embedding of each element, divide by the number of elements, and add a positional encoding. All literals in $\mathcal{I}$ have the same learned positional encoding, while all literals in $\mathcal{G}$ have a different learned positional encoding. Each action in π has a different sinusoidal positional encoding.

The Transformer encoder takes in a sequence of embeddings for π, $\mathcal{G}$, and $\mathcal{I}$, in that order. We keep only the first position of the output and add linear and Sigmoid layers to produce binary predictions. The loss function is binary cross entropy loss between the prediction and actual plan feasibility, *i.e.* whether the planner found continuous values that satisfy all the constraints posed by each action in the plan. For each batch, we add multi-headed attention to enable each position in the sequence to attend to each other, except for the action tokens, which use a causal mask.

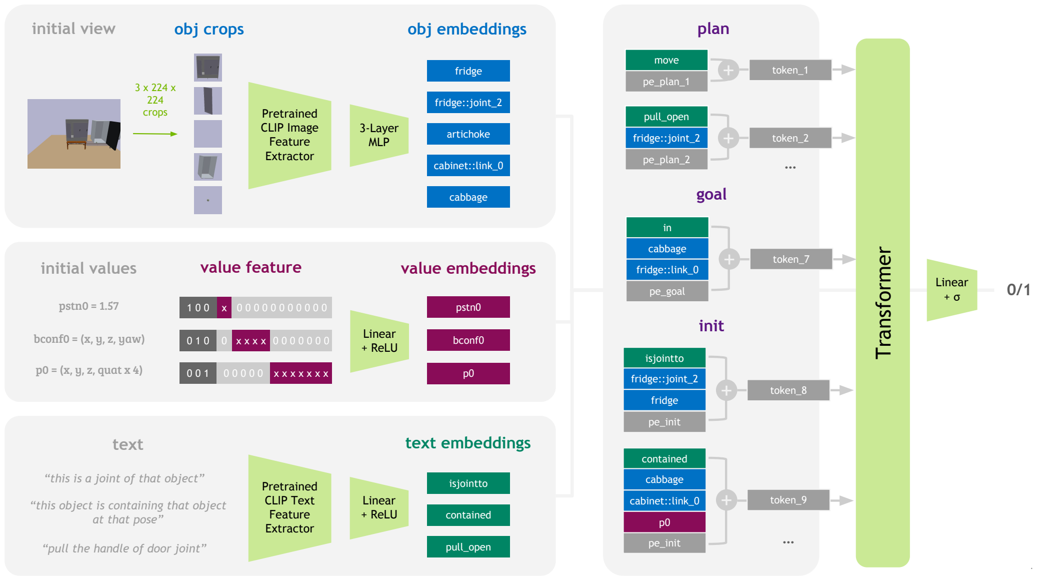


Figure 2: Architecture diagram of our PIGINet model

5 PLANNING ALGORITHM

We developed a new PDDLStream [20] algorithm that uses our PIGINet for feasibility prediction by feeding it hundreds of candidate skeletons for sorting. Similar to the *focused* algorithm [20], it optimistically instantiates a set of free action parameters X using the available sampling operations. It repeatedly searches for k distinct plan skeletons to make up a single skeleton batch Π_k, where $k \geq 1$ is a hyperparameter. New plans are identified by performing a forward plan-space search FORBID-SEARCH that forbids any previously identified plans Π from the search's open list [21]. After k attempts, the batched plans Π_k are scored using the learned feasibility predictor $f(\mathcal{I}, \pi, \mathcal{G})$. Plans that are predicted to have feasibility < 0.5 are discarded, except when all plans in the batch as below the threshold. The rest are sorted in decreasing order of predicted likelihood of feasibility. The algorithm attempts to refine each plan in order using sampling via SAMPLE-PLAN. The first fully-bound plan π_* is returned as a solution. Algorithm 1 gives the pseudocode for the main algorithm BATCH-SORTED-TAMP.

6 EXPERIMENTS & RESULTS

We carried out experiments to answer the following three questions about the model: (1) **Efficiency**: Can PIGINet improve planning speed without sacrificing planner success rate? (2) **Generalization**: Can a model trained on some problems perform well in other problems? (3) **Ablation**: Can the model still perform well with less input encoding?

Algorithm 1 Batch Sorted TAMP Plan Prediction

```
Require: Feasibility predictor: $f(\mathcal{I}, \pi, \mathcal{G}) \rightarrow [0, 1]$
 1: procedure BATCH-SORTED-TAMP($\mathcal{O}, \mathcal{I}, \mathcal{G}, \mathcal{A}; k$)
 2:     $X \leftarrow \emptyset$                                          ▷ Initialize free parameters
 3:     $\Pi \leftarrow \emptyset$                                          ▷ Initialize identified plans
 4:     while True do
 5:         $\Pi_k \leftarrow \emptyset$                                    ▷ Initialize batch of at most $k$ plans
 6:         for $i \in \{1, ..., k\}$ do
 7:             $\pi \leftarrow$ FORBID-SEARCH($\mathcal{O}, X, \mathcal{I}, \mathcal{G}, \mathcal{A}; \Pi$)        ▷ Search but forbid plans $\Pi$
 8:             if $\pi \neq$ None then                             ▷ Identified a new plan skeleton
 9:                 $\Pi \cup= \{\pi\}; \Pi_k \cup= \{\pi\}$                   ▷ Update the plan skeleton batches
10:             else                                             ▷ Infeasible search subproblem
11:                 $X \cup=$ NEW-PARAMETERS($\mathcal{O}, \mathcal{I}$)              ▷ Add more free parameters
12:         $P = \{\langle f(\mathcal{I}, \pi, \mathcal{G}), \pi\rangle \mid \pi \in \Pi_k\}$                 ▷ Scored plan skeletons
13:         for $\langle p, \pi\rangle \in$ reversed(sorted($P$)) do        ▷ Process in decreasing order of score
14:             if $p > 0$ then                                   ▷ Process if positive score
15:                 $\pi_* =$ SAMPLE-PLAN($\pi; \mathcal{I}, \mathcal{G}$)            ▷ Attempt to bind the plan skeleton
16:                 if $\pi_* \neq$ None then                      ▷ Successfully bound the skeleton
17:                     return $\pi_*$                              ▷ Return the bound solution
```

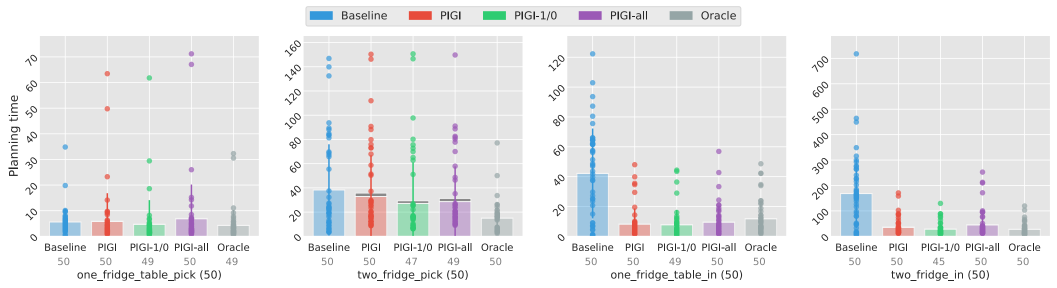
Dataset In our experimental domain, there are fridges on top of tables and food items inside the fridges or on the table. Each door fridge is closed 50% of the time or in a position sampled uniformly at random across its joint limits. We also randomly sample the pose of objects, the height of tables, and the initial base configuration of a PR2 robot. There are one or two instances of each object in each scene. The goal is to hold the food, place the food on the table, or place the food in the fridge.

After sampling a scene and a goal, we run the new planning algorithm to generate diverse skeletons for the problem and one solution to produce positive labels. Then we render the segmented images in PyBullet before training and testing. We save the *problem* $\langle \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$ along with the skeletons and images as one data point. In this way, we generated a dataset of 4,620 problems for training (with a 9-to-1 train/val split) and an additional 50 problems for testing integration with planners for each task. Depending on whether the initial pose of the food, the door handle, and the destination region is reachable, the robot may need to open ten to four doors in order to accomplish the task, resulting in minimum plan lengths ranging from two to twelve actions. We used articulated objects from PartNet-Mobility dataset [22] and convex food objects sourced online. We used nine models of fridges, eleven models of food items, and ten models of tables. They differ in shape, size, color, and texture, as shown in Figure 1.

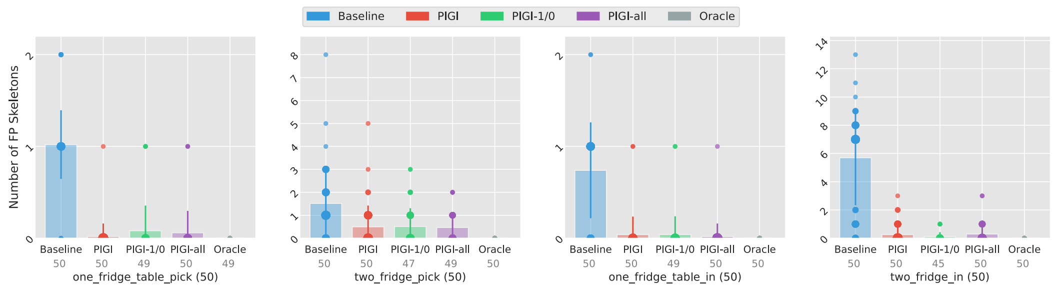
During training, we augment the images with random crops, rotations, shifts, warps, color jittering, blurring, and grayscale. We used a pre-trained CLIP model [19] for image and text embedding. For efficiency of training Transformers, we used a max sequence length of 32 and truncate extra input tokens in $\mathcal{I}$ after randomly shuffling them.

Planner performance: First, we investigate the effectiveness of PIGINet for speeding up planning. We expect the benefit of sorting skeletons based on feasibility prediction to be more pronounced for longer-horizon plans, e.g., when the goal cannot be achieved by a single pick or place, such as when moving objects table-to-fridge (`one_fridge_table_in`) or fridge-to-fridge (`two_fridge_in`). We compared these two pick-and-place problems with two single-pick problems, in a scene with either (1) one fridge and a table (`one_fridge_table_pick`) or (2) two fridges (`two_table_pick`). We used 50 problems for each of the four tasks and ran the planners with a 300-second timeout (the final refinement phase may take longer).

We compared the planning time of five different planner ablations of Algorithm 1. `Baseline` is a learning-free planner that always returns $f(\mathcal{I}, \pi, \mathcal{G}) = 1$, attemping to refine every skeleton in the order of ascending plan length. `PIGI` sorts the plans with the probability generated by our PIGINet after Sigmoid. `PIGI-0/1` uses PIGINet as a binary classifier to reject plans and attempting the accepted ones in ascending order of plan length. `PIGI-all` is trained on all tasks in the dataset



(a) The runtime of PIGINet-enabled planners compared to a non-learning `Baseline` and a clairvoyant `Oracle`.



(b) The number of false positive skeletons refined. The scattered dot sizes are proportional to their log density.

Figure 3: Evaluation of classification models. The number in bracket indicates the number of problems in the test set, while the grey number under each bar indicates the number of problems solved by the planner. Note that each subplot has a different y-axis.

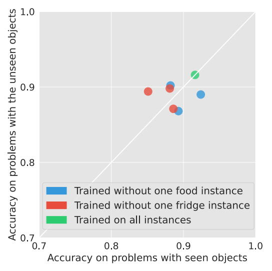
instead of just the task tested. `Oracle` refines only one plan skeleton that's already logged to be feasible offline, severing as an upper bound on possible performance.

Figure 3(a) shows that PIGINet is able to cut down planning time substantially, especially in longer-horizon problems ($\downarrow 80\%$). Given 50 or 100 plan skeletons, the model usually cuts down the number of infeasible plans to 1 - 4 and finds a solution after sampling one or two false positive skeletons, as shown in Figure 3(b). Note that the improvement on the number of false positive skeletons is larger than that on planning time for the second task `two_table_pick`. This disparity is because the shorter plans prioritized by the `Baseline` planner imposes less constraints thus needs less time to be proven infeasible during refinement than those ranked top by PIGINet, which have similar lengths to the True Positive plans. `PIGI-0/1` sometimes failed to find a plan because of its hard cutoff, while `PIGI` has a way to recover from prediction error by still attempting skeletons with prediction scores < 0.5 but are ranked high.

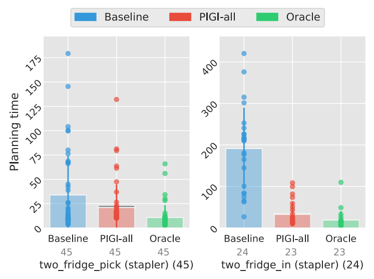
Generalization over geometry: Next, we held out one instance of an object during training and tested the model on a test set with that instance appearing in each example. We experimented with three food instances and three fridge instances. Figure 4(a) shows that the models have equally good prediction accuracy on the unseen object instances as the seen ones. We also created a test set where the goal is to rearrange staplers instead of food items. Five different stapler models from PartNet-Mobility dataset were used. We see similar improvements like those in Figure 3, showing that the models can generalize to similar problems with different categories and shape of objects.

Learner ablations: Finally, we studied how the prediction accuracy on the test set is affected when we remove different components of the multi-modal encoding.

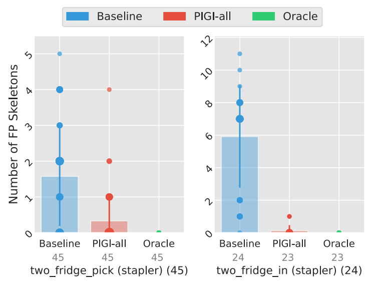
To test object embeddings, we experimented with pre-trained image features (with fine-tuning), randomized one-hot encoding, or these two added element-wise. Without the added one-hot encoding, multiple objects that are occluded will have the same image features because no pixels will be observed in the segmented crop. Figure 5(a) shows that PIGINet with image features performs slightly better than that without, while adding one-hot identifiers does not contribute much.



(a) Leave-one-out accuracy.



(b) Testing runtime on the stapler env.



(c) Testing FP count on the stapler env.

Figure 4: Evaluating the model's generalization on unseen objects.

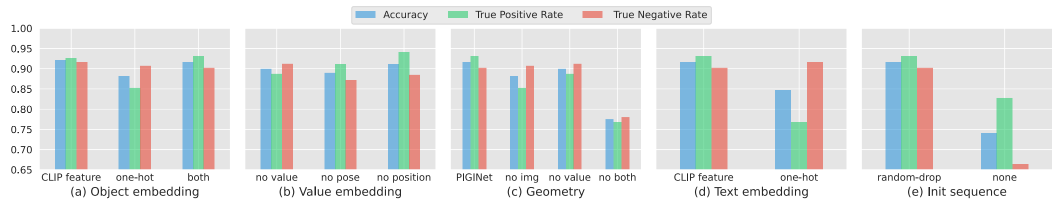


Figure 5: Classification performance on the test set across ablations of our method in multi-modal embedding and sequence composition schemes.

To test value embeddings, we experimented with removing all values, removing just object pose values, and removing just door position values. Figure 5(b) shows that removing value embeddings does not affect performance significantly.

However, if we remove both value embeddings and image features, the performance is severely compromised, as shown in Figure 5(c). We suspect that **image features of objects provide redundant information as the continuous values for representing the geometric state of the world**.

To test text embeddings, we experimented with using pre-trained CLIP text features or a fixed one-hot encoding. Figure 5(d) shows that CLIP features improved performance. We also experimented with dropping all initial literals from the sequence input. As a result, performance suffered, as shown in Figure 5(e).

7 CONCLUSION

To speed up TAMP planning, we developed a novel learning architecture, PIGINet, that encodes initial states, goal, and candidate plans to predict plan feasibility. The model is shown to reduce planning time on long-horizon rearrangement problems with articulated obstacles, where uninformed planners suffer from wasted refinement efforts. PIGINet also achieves zero-shot generalization to unseen movable object categories thanks to its visual encoding of objects.

Acknowledgments

We would like to thank Weiyu Liu, Wentao Yuan, and Danfei Xu for helpful discussions on the project.

References

[1] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Perez. Integrated Task and Motion Planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4, 2021.

[2] M. Katz, S. Sohrabi, and O. Udrea. Top-quality planning: Finding practically useful sets of best plans. In *AAAI*, volume 34, 2020.

[3] T. Ren, G. Chalvatzaki, and J. Peters. Extended tree search for robot task and motion planning. *arXiv preprint arXiv:2103.05456*, 2021.

[4] J. Ortiz-Haro, E. Karpas, M. Toussaint, and M. Katz. Conflict-directed diverse planning for logic-geometric programming. 2022.

[5] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. An incremental constraint-based framework for task and motion planning. *IJRR*, 37(10), 2018.

[6] C. R. Garrett. *Sampling-Based Robot Task and Motion Planning in the Real World*. PhD thesis, Massachusetts Institute of Technology, 2021.

[7] T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. In *AAAI*, volume 35, 2021.

[8] M. Khodeir, B. Agro, and F. Shkurti. Learning to search in task and motion planning with streams. *arXiv preprint arXiv:2111.13144*, 2021.

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

[10] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *2020 ICRA*. IEEE, 2020.

[11] L. Xu, T. Ren, G. Chalvatzaki, and J. Peters. Accelerating integrated task and motion planning with neural feasibility checking. *arXiv preprint arXiv:2203.10568*, 2022.

[12] S. A. Bouhsain, R. Alami, and T. Simeon. Learning to predict action feasibility for task and motion planning in 3d environments. 2022.

[13] S. Park, H. C. Kim, J. Baek, and J. Park. Scalable learned geometric feasibility for cooperative grasp and motion planning. *RA-L*, 2022.

[14] B. Kim and L. Shimanuki. Learning value functions with relational state representations for guiding task-and-motion planning. In *CoRL*. PMLR, 2020.

[15] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. In *ICRA*. IEEE, 2021.

[16] W. Liu, C. Paxton, T. Hermans, and D. Fox. Structformer: Learning spatial structure for language-guided semantic rearrangement of novel objects. In *2022 ICRA*. IEEE, 2022.

[17] V. Blukis, C. Paxton, D. Fox, A. Garg, and Y. Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *CoRL*. PMLR, 2022.

[18] W. Yuan, C. Paxton, K. Desingh, and D. Fox. Sornet: Spatial object-centric representations for sequential manipulation. In *CoRL*. PMLR, 2022.

[19] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*. PMLR, 2021.

[20] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. PDDLStream: Integrating Symbolic Planners and Blackbox Samplers. In *ICAPS*, 2020.

[21] C. Garrett, M. Katz, T. Lozano-Pérez, L. Kaelbling, and S. Sohrabi. Diverse planning to cover planning-time uncertainty about initial state. 2020.

[22] F. Xiang, Y. Qin, K. Mo, Y. Xia, H. Zhu, F. Liu, M. Liu, H. Jiang, Y. Yuan, H. Wang, et al. Sapien: A simulated part-based interactive environment. In *CVPR*, 2020.