2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

On Learning Scene-aware Generative State Abstractions for Task-level Mobile Manipulation Planning

Julian Förster[1], Jen Jen Chung[2], Lionel Ott[1] and Roland Siegwart[1]

***Abstract*— Task and motion planning (TAMP) is a promising approach for efficient long-horizon manipulation planning, which is a prerequisite for being able to deploy manipulation systems in human-centered environments at scale. TAMP systems often rely on so-called predicates to abstractly describe the world. Today, predicates and their groundings are often hand-engineered. Furthermore, robot action parameterizations required to fulfill desired predicates are typically discovered by sampling naively or using oracles (again hand-engineered). We aim to automate predicate discovery and grounding with a system that learns to classify the state of predicates in a set of scenes while concurrently learning to generate scene configurations that fulfill the desired predicates. Our results show that high classification accuracies and generation success rates can be achieved with architectures based on multi-layer perceptrons (MLPs) and graph neural networks (GNNs) that are trained on bounding box as well as point cloud-based features in a Generative Adversarial Network (GAN)-inspired fashion, decisively outperforming both decision tree and uniform sampler baselines. The integration of our framework into a TAMP system demonstrates its positive impact on solving mobile manipulation tasks. A reference implementation of our method and data are available at https://github.com/ethz-asl/predicate_learning.**

I. Introduction

A major motivation for developing mobile manipulation systems is to automate dangerous, tedious, dull, expensive, or inconvenient tasks that to date have to be carried out by humans in factories, businesses, homes, construction sites, or hospitals. However, most manipulation systems have been confined to structured environments, where pre-programming behaviors is viable. This in turn limited their use to tasks involving many repetitions and with a high load factor, mainly for economic reasons. Only due to advances in recent years in perception, compliant control, as well as improved cost efficiency have we opened a perspective on using mobile manipulation systems (MMS) in less structured environments.

With MMS moving into unstructured environments, the expectations shift from the execution of few tasks, many times each (e.g. robot in a factory line), to the execution of many tasks, few times each (e.g. agile manufacturing lines in the small to medium industry or assistive robots in human-centered environments). This renders pre-programming a robot for all tasks and situations it might encounter infeasible. Instead, we envision specifying goals on an abstract level and having the robot autonomously determine and execute the steps required to reach these goals [1].

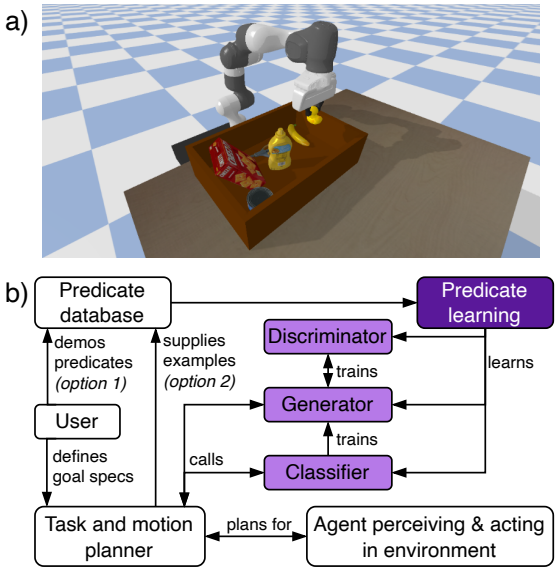

Fig. 1. Application of the proposed system: a) A MMS places an object to fulfill the "inside" predicate. b) Schematic overview of the envisioned framework: the proposed predicate learning method draws from demonstrations (optionally from a user, or directly from the agent's interactions) to learn generators and classifiers, which can be used by downstream planning algorithms for observing the state and parameterizing actions such that user-defined goal predicates are fulfilled. The components this work focuses on are shown in purple.

Conventional motion planning for achieving such multi-step long horizon goals in high-dimensional domains quickly becomes computationally intractable, especially for systems with many degrees of freedom operating in domains with many objects and disturbances. This can be addressed by resorting to higher level planning systems as developed by the symbolic planning [2] and task and motion planning (TAMP) [3], [4] communities. The added layer of abstraction by higher-level planners further introduces the benefit of an improved human-robot interface. Instead of specifying low-level goals for motion planners directly, desired goals are specified using binary abstract state variables – typically referred to as *predicates* – and the planner plans towards achieving them. The role of predicates is twofold. First, they constitute the state abstractions used to specify a goal (e.g. "plate in dish washer"). Second, predicates are used to model preconditions and effects for robot skills, thus constituting the state variables used by planners to produce correct and successful plans.

This work was supported in part by the European Union's Horizon 2020 research and innovation programme under grant agreement No 955356.

[1]Julian Förster, Lionel Ott and Roland Siegwart are with the Autonomous Systems Lab, ETH Zurich, Switzerland {fjulian,lioott,rsiegwart}@ethz.ch

[2]Jen Jen Chung is with the The University of Queensland, St. Lucia, Australia jenjen.chung@uq.edu.au

Eventually, task plans are to be executed in the real world. For this, predicates need to be linked to the potentially high-dimensional states they represent. We refer to this link as *grounding* and for our purposes, a predicate grounding needs two components:

- *Classification*: to decide whether a predicate holds given the state of the agent and its environment;
- *Generation*: to set the action parameters (e.g. grasp configurations, placement poses, navigation target coordinates, etc.) that result in a predicate being satisfied.

Today's planning systems mostly rely on hand-engineered model-based predicate groundings, especially for the generation component [5], which severely limits their flexibility and ease-of-use. With the goal of using MMS in open-world settings, meaning that the set of scenarios a robot might encounter is not closed and known at design time, new predicates will need to be added as new situations and tasks arise. In this context, manually designing predicate groundings is tedious and requires an expert, thus preventing quickly scaling to new scenarios. Furthermore, manually designed groundings tend to use simple geometric features of objects of interest (positions, bounding boxes, etc.), as using raw geometric representations like point clouds is challenging. This makes manually modelling predicates whose interpretation relies on intricate geometric details of the target objects very difficult.

In this work, we attempt to address this limitation by learning the two components of predicate groundings (classification and generation) from data, as visualized in Fig. 1. We envision the required training data either to be provided by user demonstrations (e.g. through a graphical user interface (GUI) as demonstrated in the accompanying video) or to be collected in a self-supervised manner by the robotic agent. This will allow flexibly adapting MMS to new scenarios, greatly enhancing the range of tasks they can be applied to and thus making progress towards the vision of automation in unstructured environments. The contribution of this work is threefold:

1) A framework for learning predicate groundings, consisting of a classifier and a generator component to generate target configurations that fulfill a given predicate,
2) A GNN-based architecture that takes surrounding objects into account for classification and generation,
3) Integration of the proposed system into a complete TAMP pipeline and evaluation in simulation, showcasing how the method can be used in a mobile manipulation application.

II. Related Work

With the motivation to facilitate and speed up task-level planning, several past works proposed methods for learning state abstractions. A prominent line of work focuses on learning to classify a pre-defined set of spatial predicates based on labeled data, for example focusing on relations between two objects at a time [6]. Leveraging a different source of supervision [7], symbols can be learned in a robot's observation space and grounded to language instructions for a downstream instruction-following planner. The method of James et al. [8] results in similar state abstractions. However, they are extracted in a self-supervised fashion while the robot is exploring the environment and its action space. For new environments, a fine-tuning phase is necessary to make the abstractions applicable. Instead of leveraging perception information as input, the abstractions can be based on continuous feature variables (e.g. object positions), which are partitioned using landmarks (change in a variable leading to a qualitative difference in a scene) [9], [10].

None of the works cited above leverage a predicate grounding to parameterize actions (i.e. the generative component of a grounding we refer to in Sec. I). To our knowledge, only few such approaches exist; this is despite advances in GANs [11] and Variational Auto-Encoders (VAEs) [12] showing impressive results in other fields. A method proposed by Kim et al. [13] aims to learn a GAN-based generator for sampling action parameterizations in the context of TAMP. For training, uniformly sampled parameters are used in conjunction with task success as supervision signal. The method ignores other objects in the scene as well as scene and object geometry. In addition, since it is retrained for each task, the resulting samplers do not generalize to other tasks or environments. CLIPORT [14] predicts placement poses being trained on few demonstrations (50 to 100). However, it is limited to table top scenes and requires days for training, preventing ad-hoc learning of new predicates in interaction with users. Kase et al. [15] combine models to predict predicate states and robot arm joint positions with a symbolic planner. The focus on planning yields impressive reactive execution capabilities. However, training the models requires a hand-engineered expert policy with full access to robot and environment state. Scene grammars [16] provide a compelling way to describe, parse and sample from distributions over scenes. However, the types of objects involved in a scene need to be fixed a priori, and object geometry cannot be taken into account. Closely related to our work, Chitnis et al. [17] learn neural samplers to parameterize symbolic actions within a TAMP framework, however, assuming that predicates are given and ignoring object geometry.

While recent approaches building on large language models for planning have shown impressive results [18], we believe that approaches building on symbolic planning are modular, light-weight, interpretable and easy to extend/adapt and will thus play an important role in the future of MMSs.

Encouraged by the successes of these related approaches, our work aims to improve on the current state of the art for predicate grounding by learning a classifier and a generator simultaneously, as data-efficiently as possible to allow for user demonstrations, with an architecture that takes surrounding objects as well as object geometry into account to cover a wider range of predicates.

III. Learning Predicate Groundings

We propose a learning framework consisting of three models that jointly address the classification and generation

components of predicate grounding (see Fig. 1). The *classifier* takes features of scene objects as input and judges whether the predicate it is trained for holds or not. The *generator* takes the same input with the goal to produce modified features for the target objects that would make the predicate *true*. To train the generator, we take inspiration from GANs [11]. A *discriminator* is trained together with the generator to distinguish between generated scenes and scenes from the dataset. Its sole role is to provide a training signal to the generator, and it is not used after training.

A. Targeted Predicates

The primary focus of this work are predicates that describe arbitrary spatial arrangements of objects. For this, predicates accept a fixed number of arguments, where argument objects are the main subjects of the state described by the predicate. However, the predicate is determined not only by the argument objects but may also be influenced by any object present in the scene. For example, placing an object *inside* a cluttered region presumes a placement pose that is not in collision with any other object in this area. This motivates our decision to take non-argument objects into account for classification and parameter generation.

Our framework can be applied to learning generic predicates that are only reliant on observable spatial data. However, in this work, we focus our experiments on the *on* and *inside* predicates as they offer a natural curriculum of simple to complex scenarios. The simplest *on* scenario involves only the supported and supporting objects and applies few constraints on the arrangement between the two objects since only the relative positions are critical. The *inside* predicate increases complexity by introducing orientation constraints since a tall object could be *inside* a drawer if laid flat, but not if it is stood upright and exceeds the sides of the drawer. Varying the degree of scene clutter for either predicate further constrains the valid set of parameterisations. The same process can be applied to learn more complex predicates such as *stack* or *pack*, which place even greater constraints on object pose (e.g. to maintain a stable stack) and may involve more argument objects.

B. Object-centric Scene Features

Since this work focuses on spatial relations between objects, we opt for an object-centric approach of one feature vector per object. Two alternative feature-vector representations were investigated: *bounding box (BB)*-based ones, and *point cloud (PC)*-based ones. Our method is not limited to these features and could be extended to other object-centric features. However, we believe that many predicates can be modeled using the features we describe below. As in related work (e.g. [13], [17], [9]), both formulations rely on a perception pipeline for scene segmentation.

1) BB features: Our BB-based features are composed of:

Centroid position of an object-aligned bounding box.

Orientation expressed with respect to the global frame, encoded as a quaternion.

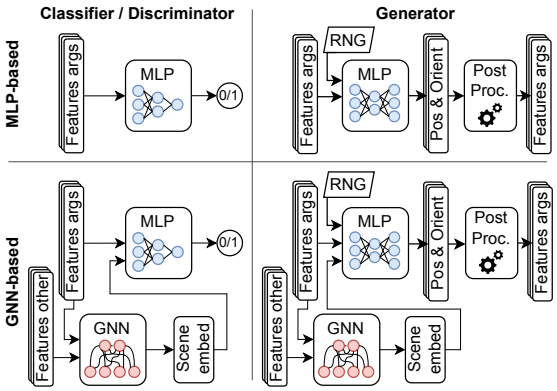

Fig. 2. MLP-based and hybrid GNN-based models used to learn predicate groundings. The classifier and discriminator share the same model architecture but are trained separately. "RNG" stands for *random number generator*.

Axis-aligned bounding box (AABB) expressed in the global frame, encoded as two opposite corner points.

Object-aligned bounding box being the smallest possible bounding box, aligned with an object's principal axes, expressed in the respective object's local frame and also encoded as two opposite corner points.

While the AABB contains information about an object's presence in the scene, the object-aligned BB more closely represents the object's size.

2) PC features: These features are obtained from an encoder that is trained in an auto-encoder scheme. We investigated several architectures including PointNet++ [19], 1D convolutions [20], and a fully connected architecture, trained on either our own dataset of 3D objects (see Sec. III-E) or on ShapeNet data [21]. We also investigated different strategies for scaling the object point clouds before encoding:

(a) Scale all point clouds with the same factor, such that the largest extent of the largest object in the training set fits inside a cube of defined size (i.e. all point clouds lie within this cube).

(b) Scale all point clouds individually, such that the largest extent of each point cloud lies on the cube (i.e. all point clouds have the same size).

The best downstream classification performance was achieved when training on ShapeNet for an encoding size of 128 with a 1D convolution-based encoder, a fully connected decoder and scaling option (a) with the target cube spanning $[-0.5, 0.5]$ on each axis.

Since the point cloud encoding only encodes information about object geometry, and not the object's pose in the scene, the final PC feature vector is composed as follows:

Centroid position of an object point cloud.

Orientation expressed with respect to the global frame, encoded as a quaternion.

Point cloud encoding as obtained from the encoder.

C. Classifier, Discriminator and Generator Architectures

We evaluated two different model architectures for the classifier, discriminator, and generator: an MLP-based architecture, which only considers the argument objects, and a hybrid GNN-based architecture, which can take into account

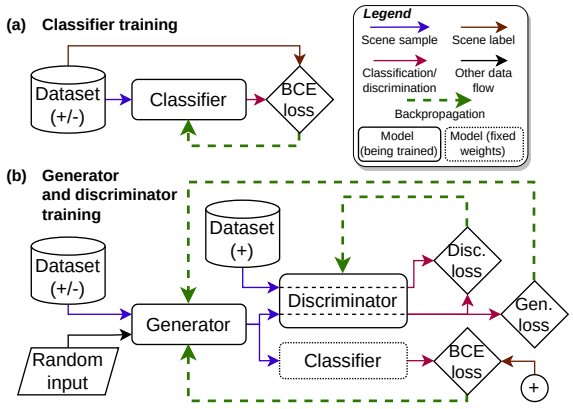

Fig. 3. Training procedure for learning predicate groundings. (a) First, the classifier is trained. (b) Then, with its weights frozen, it is used together with the discriminator to train the generator. For datasets, (+/-) stands for positive and negative samples, and (+) stands for positive samples only.

all scene objects (see Fig. 2). The classifier and discriminator share the same architecture but are trained separately.

The MLP-based models consist of fully connected layers in conjunction with ReLU activations after every layer apart from the last one. For the classifier (and discriminator), the final 1D output is obtained through a softmax layer. In contrast, the generator must output a complete set of feature vectors for downstream discrimination and classification. Thus, we condition the generator's MLP on the current scene features and task it to propose a centroid position and orientation for the argument object such that the predicate would hold. A post-processing step then completes these into a full feature vector using the input object's PC or BB features.

MLPs have the drawback that the input size is fixed, making it difficult to handle varying numbers of objects in a scene. Thus, the MLP only takes the features of the argument objects of the predicate as input. To incorporate scene-level information, we propose a hybrid GNN model architecture that captures information from surrounding objects which may influence the predicate state.

The GNN architecture adds an additional set of fully connected layers that are used to compute a scene embedding. As input, the scene embedding module takes the argument objects' feature vectors, as well as the feature vector of a surrounding object. The output is an intermediate embedding. After applying this once for each surrounding object, all intermediate embeddings go through mean pooling, making the resulting scene embedding permutation invariant w.r.t. the order of surrounding objects. Finally, the argument objects' features and the scene embedding are concatenated and fed through an MLP that is similar to the MLP-based architecture described above. The only difference being a larger input layer to accommodate the scene embedding. Post processing is added in the same way as for the MLP-based models.

D. Training Procedure

Training procedures for the proposed models are shown in Fig. 3. Since the classifier's task (classification in contrast to generation, see Sec. I) is different from generator and discriminator, it is trained separately from the other models. We train the classifier using a standard binary cross-entropy loss:

$$L_C = -\left(y \cdot \log(C(\boldsymbol{x})) + (1-y) \cdot \log(1-C(\boldsymbol{x}))\right), \quad (1)$$

where $\boldsymbol{x}$ is a sample from the dataset Q_d, y is the corresponding label and C is the classifier model.

The generator and discriminator training is inspired by typical GAN training procedures. During training, we only present positive samples from the dataset to the discriminator. This is because the adversarial training scheme causes the generator's output distribution to approximate the distribution of dataset samples passed to the discriminator. Therefore, passing positive samples only to the discriminator trains the generator to produce new samples for which the respective predicate holds. In contrast, the generator is seeded with both positive and negative samples and must learn to correctly adjust the object features from negative samples, while allowing positive samples to pass through unedited (or edited in such a way that the predicate still holds).

Both discriminator and generator are trained using a Wasserstein loss [22] together with a gradient penalty [23]. With this, the discriminator loss becomes,

$$L_D = \overbrace{-\mathop{\mathbb{E}}_{\boldsymbol{x}\sim Q_d^+}[D(\boldsymbol{x})]}^{\text{dataset samples}} + \overbrace{\mathop{\mathbb{E}}_{\boldsymbol{x}\sim Q_d, \boldsymbol{z}\sim\mathbb{N}}[D(G(\boldsymbol{x},\boldsymbol{z}))]}^{\text{generated samples}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim Q_{\hat{\boldsymbol{x}}}} \left(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1\right)^2}_{\text{gradient penalty}}, \quad (2)$$

where D and G are the discriminator and generator model respectively, $D(\boldsymbol{x})$ the discriminator's output for input $\boldsymbol{x}$, Q_d the dataset, Q_d^+ the subset of the dataset containing only positive samples. $\mathbb{N}$ is a distribution (e.g. normal or uniform), λ the gradient penalty coefficient, $\hat{\boldsymbol{x}} = \varepsilon\boldsymbol{x} + (1-\varepsilon)\tilde{\boldsymbol{x}}$, where $\boldsymbol{x} \sim Q_d^+$ a dataset sample, $\tilde{\boldsymbol{x}} \sim G(\boldsymbol{x})$ a generated sample, $\varepsilon \sim U[0,1]$, U being a uniform distribution. The generator loss when trained in conjunction with the discriminator becomes,

$$L_G^{(D)} = -\mathop{\mathbb{E}}_{\boldsymbol{x}\sim Q_d, \boldsymbol{z}\sim\mathbb{N}}[D(G(\boldsymbol{x},\boldsymbol{z}))]. \quad (3)$$

Wanting the generator to output samples the classifier labels as fulfilling the target predicate gives rise to the idea to use the classifier as an alternative training signal. The classifier learns to label scenes according to their conformity with the target predicate. We employ a binary cross-entropy loss where the label is always "true", motivating the generator to output samples the classifier labels as fulfilling the target predicate:

$$L_G^{(C)} = -\log(C(G(\boldsymbol{x},\boldsymbol{z}))). \quad (4)$$

As we have two models to provide feedback to the generator, we can formulate three different generator training schemes:

Discriminator only where $L_G = L_G^{(D)}$, and discriminator and generator are trained simultaneously.

Classifier only where $L_G = L_G^{(C)}$. After training the classifier, its weights are frozen and it is used as the sole objective during generator training.

Both where $L_G = L_G^{(D)} + L_G^{(C)}$, i.e. the generator is trained with feedback from the classifier and the discriminator at the same time. The classifier's weights are trained a priori and are frozen during generator training, while the discriminator is trained alongside the generator.

E. Training Data

As mentioned in Section III-A, we develop and test our algorithm using two predicates: `on_clutter` and `inside`. The `on_clutter` predicate takes two main arguments, a supporting object and a supported object, returning *true* only if the supported object rests stably on top of the supporting object, neither penetrating the supporting object nor colliding with any other object. The `inside` predicate also takes two main arguments, a containing object and a contained object, returning *true* if the contained object is fully inside the containing object, neither protruding from any opening, nor penetrating the containing object or other objects in the scene.

Our vision is to enable learning new predicates from non-expert user demonstrations or in a self-supervision fashion via exploration in a back-end physics simulator [24]. For the former, we developed the prototype GUI (shown in accompanying video) which allows a user to manipulate objects in a physics simulation to easily provide demonstrations.

For this work however, to reliably obtain sufficient and consistent data for evaluating our proposed method, we algorithmically generate data. For `on_clutter`, we employ the targeted sampler from iGibson 2.0 [25]. As supported objects, we use the items from the YCB dataset [26]. As supporting objects, we use items from iGibson and pybullet [27]. The generated dataset is balanced, i.e. half of the samples are positive (predicate holds) and half of the samples are negative (predicate does not hold). For each sample, a supporting and a supported object is sampled. In addition, further distractor objects are spawned at random positions and orientations. Positive $(+)$ and negative $(-)$ samples are then obtained with a pre-defined distribution over the following seven conditions.

1) $(+)$ Supported object **alone** on supporting object.
2) $(+)$ Supported object on supporting object, **together** with one or more distractor objects.
3) $(-)$ Supported object spawned at random pose, then either left **afloat** or dropped to **ground**.
4) $(-)$ Supported object placed onto supporting object, then randomly shifted up or down in z direction so that it **floats above** or is **sunken into** the supporting object.
5) $(-)$ Supported object placed close to the centroid, **inside of** the supporting object.
6) $(-)$ Supported object is not placed onto the supporting object, but onto a **different supporting object**.
7) $(-)$ Supported object placed on supporting object such that it is **in collision** with another object that rests on supporting object.

Apart from this, all object scales are randomly varied between samples. Overall, the dataset consists of 20'000 examples intended for training, and additional 2'000 examples that were generated with objects not included in the training set, intended for testing.

Training data for `inside` is generated with a custom implemented targeted sampler. Objects, dataset composition, size, separate test set, and general data generation procedure are the same as described above for `on_clutter`. The types of samples are also similar. The following lists only the differences:

- For all sample types, replace "on" and "onto" with "inside", "supported" with "contained", and "supporting" with "containing".
- No samples of type 5).
- Additional negative $(-)$ sample type: contained object is placed inside containing object, but in a way that it **protrudes** from the container.

IV. Evaluation

In the experiments, we aim to answer the following questions: (**Q1**) Can we learn predicate groundings consisting of classifiers and generators that are useful for task-level planning? (**Q2**) How data efficient is learning predicate groundings? (**Q3**) Does taking surrounding objects into account improve classifier and/or generator performance? (**Q4**) Does integrating the proposed generators into a TAMP system improve planning performance?

A. Baselines

We compare our proposed method against a range of baselines. The first baseline, which we refer to as "sampler, no class." ("class." stands for "classifier"), is based on uniform sampling. Given the argument objects, poses for the second argument object (supported, or contained respectively for the two predicates we test with) are sampled uniformly within a space around the first argument object. For the following experiments, we enlarged the BB of the first argument object by 1.5 times to obtain the sampling space. While this limits the predicates that this baseline can model, it is a necessary assumption to restrict the sampling space and thus achieve an acceptable success rate for predicates that match this assumption. Note that our proposed method makes no such assumption. Furthermore, relying on an argument object's BB would necessitate extracting BBs from PC features. Thus, we evaluate samplers on BB features only. This baseline only covers the generation component of a predicate grounding, so for deployment, a separate classifier would be required.

Building on this sampler, the baseline "sampler, MLP class." makes use of our classifier architecture described in Sec. III-C. With the basic sampling procedure identical to the previous baseline, this baseline classifies every sample and repeats sampling until a sample is classified as positive, or until a search budget is depleted. Balancing inference time with generation success rate, the budget for the experiments below is set to a maximum of 500 samples. The "sampler, MLP class." does not take into account surrounding objects

since the MLP classifier only considers the two argument objects. In contrast, the baseline "sampler, GNN class." encodes surrounding objects through the use of the GNN classifier architecture.

Further, we compared against using a decision tree (DT) for classification. In addition, to use a trained DT to generate a sample, we randomly select a leaf node that classifies positively. From there, the tree is traversed bottom up, at every node updating upper and lower bounds on the feature the node differentiates upon. Upon reaching the tree's root, this yields bounds on a subset of features. When using this baseline with the BB features, any limits on the features are converted to limits on the position of the second argument object. For the PC features, limits on the point cloud encoding are ignored. Respecting the limits on position and orientation features, we sample uniformly. The resulting generated samples are completed using the same post-processing technique applied in our proposed models (see Sec. III-C).

B. Setup and Metrics

To evaluate the proposed framework, we use the training data that was gathered as described in Sec. III-E for the two predicates `on_clutter` and `inside`, both in the form of BB features and PC features. With the goal to test the proposed method's effectiveness as predicate grounding, we are interested in the two required components of classification and generation.

We characterize classifier performance in terms of accuracy. To test generator performance, we rely on manually implemented oracles for each predicate to compute a *manual classification accuracy (MCA)* using BB features of argument and surrounding objects and a physics simulator for collision checking. Note that although hand-crafted oracles could be used to replace our learned classifiers, they are impracticable to design for every predicate encountered in real-world applications, a limitation that our proposed method aims to address. Thus, the oracles are only used for this evaluation and are not required for applying our method.

For every instance of our method, the classifier and generator are trained for 30'000 iterations each. An iteration constitutes a single batch being passed through a model with a subsequent back-propagation and weight update. For this work, we consistently use a batch size of 16. Depending on the training mode, the discriminator is trained alongside the generator, while the classifier's weights are frozen during generator training.

For methods taking surrounding objects into account (baselines as well as proposed), we select as surrounding objects all objects in a given scene that are within 1 m of any argument object. For other predicates, this threshold would potentially need to be adjusted depending on the size of objects being manipulated.

C. Results

The results from the classifier training are presented in Fig. 4. All methods achieve a high training accuracy on both

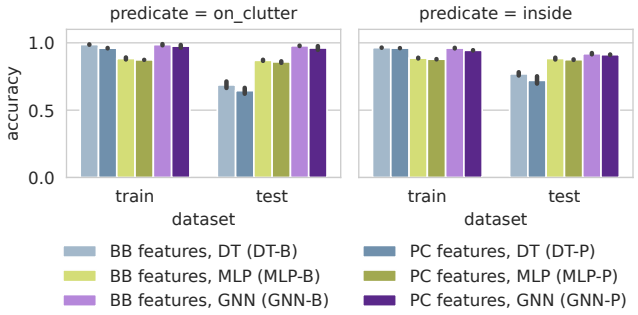

Fig. 4. Results on **classifier** accuracy. The color of the bars represents the model we used. Every method is trained for five different random seeds, the error bars represent the resulting spread. While both baselines (blue, green) and our proposed method (purple) exhibit a high training performance, the baselines' test performance is significantly reduced, hinting at overfitting.

predicates. Between the three methods, the MLP classifier shows the lowest training accuracy. This is because it does not take surrounding objects into account and tends to misclassify samples where the predicate would hold but one of the argument objects is in collision with a surrounding object (sample type 7 in Sec. III-E). While the DT baseline achieves high training accuracy, it drops significantly in test performance. This indicates overfitting to the training data. Our proposed methods instead maintain a high performance on the unseen test set, allowing us to positively answer **Q1** regarding classification. For both predicates, the difference between using BB features or PC features is marginal. However, using the GNN model, which accounts for surrounding objects, is key to increasing classification performance **Q3**.

To answer **Q2**, we analyse the generator training results in Fig. 5. In addition to the training runs on the full 20'000 samples of the training set, we trained each method on subsets of data to investigate performance in low data regimes, which would be encountered when sourcing demonstrations from users. Without limiting the training dataset, the highest performance for both predicates is achieved by our MLP-based method using PC features, achieving average MCAs of 0.77 for `on_clutter` and 0.64 for `inside`. The other variants we proposed achieve similar performances. This is in contrast to the baselines. For the `on_clutter` predicate, the best baseline is "sampler, MLP class." with an MCA of 0.11. For `inside`, "sampler, GNN class." performs best among the baselines, achieving an MCA of 0.25. For our proposed methods, training on the BB features seems to require more data while it also achieves a slightly lower performance compared to results based on PC features. Similarly, the baselines are data hungry. For both predicates, both sampler and DT generators only come close to their final performance starting from a dataset size of around 5'000 samples.

In contrast, our method applied to PC features achieves a high performance also for low sample numbers. For `on_clutter`, our MLP-based method achieves 65% and 99% of its peak performance with 100 and 500 samples, respectively. For `inside`, this is even more pronounced. Using only 10 training samples, it achieves 76% of its peak performance on the test set. While the GNN model with PC features needs slightly more samples (72% after 100 samples

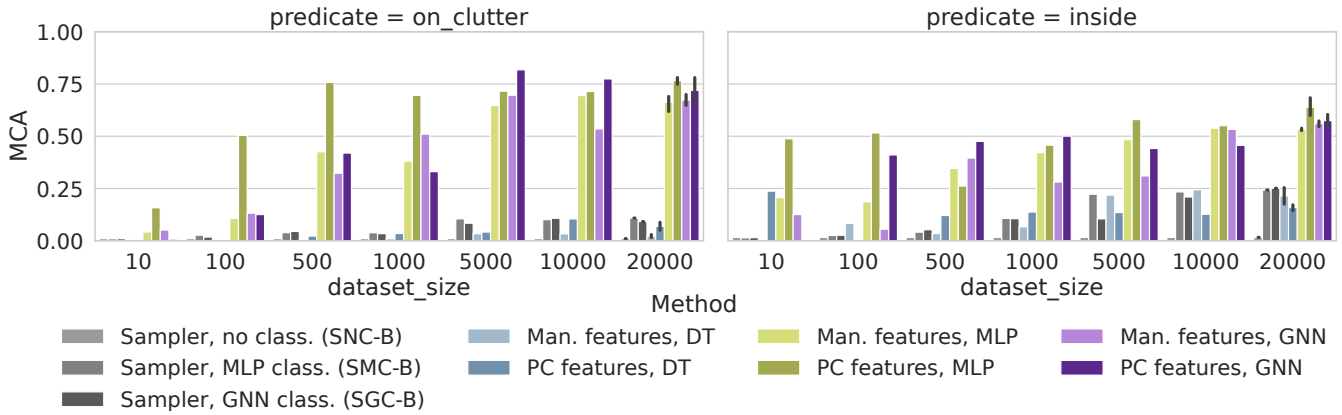

Fig. 5. **Generator** performance in terms of MCA on the test set for all baselines and our proposed methods for different training dataset sizes. The runs on the full dataset (20'000 samples) are repeated for five random seeds each, the resulting variance is visualized using error bars.

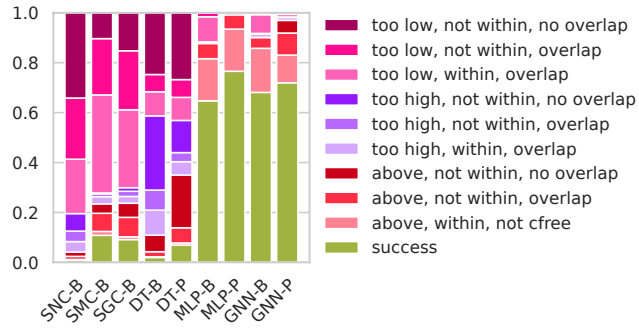

Fig. 6. Distribution of generator outputs for the `on_clutter` test dataset with generators trained on 20'000 samples, averaged over five random seeds for each method. Failure cases are classified in terms of z position as *too high/low* (*above* means success), whether the supported object is *within* horizontal bounds imposed by supporting object, if not within whether there is any *overlap* at all, and if there are collisions with other objects (*cfree*).

of `inside`), it is still ahead of its counterpart trained on BB features (71% after 500 samples).

While GNN models outperform the MLP models for classification, the MLP models often show a slightly higher performance as generators compared to GNN generators. For classification, detecting collisions between objects is important, explaining the inferior performance of the MLPs. For generation, and the predicates chosen here, awareness of surrounding objects seems to be less crucial as the MLP model still finds enough collision-free samples by chance, even without awareness of surrounding objects.

The distribution of generator outputs for `on_clutter` is shown in Fig. 6, with unsuccessful cases further separated by failure type. Failed samples from our method are often close to success, supported objects being either on the supporting object but in collision with another object, or slightly outside the horizontal boundaries ("not within") while still overlapping with the supporting object. In contrast, all baselines have significant proportions of samples that are far from success (no vertical/horizontal alignment, no overlap).

In Fig. 7, we report inference times for all methods. This is important since when embedding groundings into downstream planning operations, the planner's runtime is highly dependent on fast sampling of potential target configurations. For classification, using PC features and GNN models in-

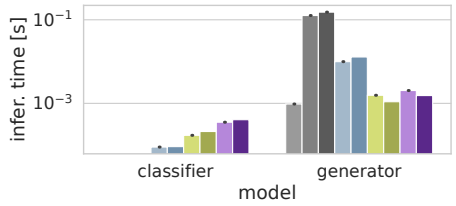

Fig. 7. Inference times for all classification and generation baselines and proposed methods. Legend, see Fig. 5. The inference times are averaged for each method over 2'000 forward passes on an *i7-9750H* CPU. Note that the sampler baselines cannot classify.

creases inference time as expected, but not prohibitively. For generation, our proposed methods show moderate inference times, while DT generators and classifier-based samplers are one and two orders of magnitude slower, respectively. Combined with its superior classification and generation performance, this further shows our proposed method's fitness to serve as predicate grounding to be used in downstream planning tasks, supporting a positive answer to **Q1**.

D. TAMP Integration

We integrated our predicate learning framework into a simple TAMP implementation to test its performance in a planning scenario where a simulated mobile manipulator is tasked with placing four distinct objects onto a cupboard. The generators are called by the planner to sample placement positions for the objects. Each configuration is run to solve the problem 100 times, with a time budget of 30 seconds per run. Fig. 8 shows that all baselines have a success rate below 50%. In contrast, while our MLP generator using BB features solves 58% of the problems, its PC feature pendant, as well as both GNN generator variants solve 100% of the runs and produce successful samples more than twice as fast compared to baselines. This highlights our methods' positive impact on TAMP success rates and planning times **Q4**.

V. Discussion and Outlook

This paper presents a framework for learning classifiers and generators for predicate groundings from demonstration data. Investigating both bounding box- and point cloud-based features, our results suggest that PC features perform superior in terms of classification accuracy and generation

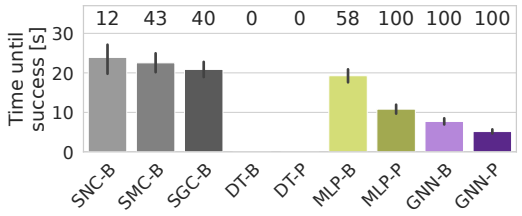

Fig. 8. Results of integrating generators into a simple TAMP system and solving a test problem (place 4 objects onto a cupboard) 100 times each. Numbers above bars indicate successfully solved tasks out of 100. The bars indicate planning time until a solution is found, only including successful runs (no bar for DT due to no successful runs).

success rate. Especially data efficiency benefits from the PC features, highlighting the advantage of including more detailed geometry information.

Further, our hybrid GNN models are capable of taking surrounding objects into account. While this mainly improves classification accuracy, generator performance is similar to the MLP counterparts when analyzed in isolation. However, once moving to the TAMP integration experiments, we observe the GNN coming in at both a higher success rate and a shorter time until success, spotlighting the advantage of taking surrounding objects into account in mobile manipulation planning scenarios.

With these properties our proposed method improves sampling goal configurations for TAMP in two key aspects: first, as shown in the experiments, our method outperforms uniform samplers (even if enhanced by classifiers) as they are typically used in existing planning systems. Second and more importantly, our method can be trained on demonstrations by non-expert users, removing the need to hand-engineer model-based classifiers and generators, and thus allowing to scale to open world application domains such as mobile manipulation in unstructured environments.

Acknowledgement

The authors would like to thank Paula Wulkop for valuable feedback on the manuscript.

References

[1] G. Ajaykumar, M. Steele, and C.-M. Huang, "A Survey on End-User Robot Programming," *ACM Computing Surveys*, vol. 54, no. 8, pp. 1–36, Nov. 2021.

[2] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. V. Weghe, "HERB: A home exploring robotic butler," *Autonomous Robots*, vol. 28, no. 1, pp. 5–20, 2010.

[3] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1470–1477.

[4] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated Task and Motion Planning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 265–293, May 2021.

[5] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Perez, "Learning Symbolic Operators for Task and Motion Planning," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 3182–3189.

[6] F. Yan, D. Wang, and H. He, "Robotic Understanding of Spatial Relationships Using Neural-Logic Learning," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020, pp. 8358–8365.

[7] N. Gopalan, E. Rosen, S. Tellex, and G. Konidaris, "Simultaneously Learning Transferable Symbols and Language Groundings from Perceptual Data for Instruction Following," in *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, 2020.

[8] S. James, B. Rosman, and G. Konidaris, "Learning Portable Representations for High-Level Planning," in *International Conference on Machine Learning*, 2020, pp. 4682–4691.

[9] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum, "Predicate Invention for Bilevel Planning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, pp. 12 120–12 129, 2023.

[10] J. Mugan and B. Kuipers, "Autonomous Learning of High-Level States and Actions in Continuous Environments," *IEEE Transactions on Autonomous Mental Development*, vol. 4, no. 1, pp. 70–86, 2012.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[12] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv:1312.6114*, 2014.

[13] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez, "Guiding Search in Continuous State-Action Spaces by Learning an Action Sampler from Off-target Search Experience," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[14] M. Shridhar, L. Manuelli, and D. Fox, "CLIPORT: What and Where Pathways for Robotic Manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 894–906.

[15] K. Kase, C. Paxton, H. Mazhar, T. Ogata, and D. Fox, "Transferable Task Execution from Pixels through Deep Planning Domain Learning," in *2020 IEEE International Conference on Robotics and Automation*, 2020, pp. 10 459–10 465.

[16] G. Izatt and R. Tedrake, "Generative Modeling of Environments with Scene Grammars and Variational Inference," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2020, pp. 6891–6897.

[17] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 4166–4173.

[18] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Task Planning," 2023, arXiv:2307.06135.

[19] C. Qi, L. Yi, H. Su, and L. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 5105–5114.

[20] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning Representations and Generative Models for 3D Point Clouds," in *International Conference on Machine Learning*, 2018, pp. 40–49.

[21] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An Information-Rich 3D Model Repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

[22] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein Generative Adversarial Networks," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, pp. 214–223.

[23] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[24] J. Förster, L. Ott, J. Nieto, N. Lawrance, R. Siegwart, and J. J. Chung, "Automatic extension of a symbolic mobile manipulation skill set," *Robotics and Autonomous Systems*, vol. 165, p. 104428, 2023.

[25] C. Li, F. Xia, R. Martín-Martín, M. Lingelbach, S. Srivastava, B. Shen, K. Vainio, C. Gokmen, G. Dharan, T. Jain, A. Kurenkov, K. Liu, H. Gweon, J. Wu, L. Fei-Fei, and S. Savarese, "iGibson 2.0: Object-Centric Simulation for Robot Learning of Everyday Household Tasks," in *Conference on Robot Learning (CoRL)*, 2021.

[26] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Yale-CMU-Berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017.

[27] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016.