Generative Modeling of Environments with Scene Grammars and Variational Inference

Gregory Izatt and Russ Tedrake
{gizatt, russt}@csail.mit.edu

***Abstract*— How do we verify that a cleaning robot that we have tested only in a simulator and in case studies in the lab, will work in every house in the world? A critical step in answering that question is to establish a quantitative understanding of the distribution of environments that a robot will face when when it is deployed. However, even restricting attention only to the distribution of *objects* in a scene, these distributions over environments are nontrivial: they describe mixtures of discrete and continuous variables related to the number, type, poses, and attributes of objects in the scene. We describe a probabilistic generative model that uses scene trees to capture hierarchical relationships between collections of objects, as well as a variational inference algorithm for tuning that model to best match a set of observed environments without any need for tediously labeled parse trees. We demonstrate that this model can accurately capture the distribution of a pair of nontrivial manipulation-relevant datasets and be deployed as a density estimator and outlier detector for novel environments.**

I. Introduction

In order to safely and successfully deploy robots at scale in the real world, we need to understand the distributional robustness of the systems we are deploying. An essential piece of that puzzle is to gain an understanding of how the world is distributed. Even when restricting attention merely to the distribution of *objects* in a scene, this distribution is quite complicated: objects vary discretely in number and type, and each object itself varies in pose, shape, material, etc. All of these parameters may covary: for example, many classes of objects may only ever appear in pairs or sets, and members of that set may match in e.g. color.

A model that can capture these distributions would be extraordinarily useful – not only could it generate enormous amounts of relevant training data for simulation-based testing and verification, but it could be used to detect when out-of-distribution environments are encountered at runtime. To be useful for these ends, however, the model must be very accurate – and to be informative for the engineers of robot systems, the models should be transparent as to why they produce the scores they do.

We present a step in the direction of achieving informative and flexible generative models for these kinds of environments. We use scene grammars and the trees they describe as a core framework to break down the dependency structure of complex environments. A scene tree explains a scene in terms of a hierarchy of instances of *symbols*: atomic object instances are represented by instances of terminal symbols at the leaves of the tree, while groups of objects are explained by intermediate non-terminal symbol instances. (See Figure 1 for some examples.) Typically, these intermediate symbols represent semantically meaningful groupings of objects: for example, place settings (as groupings of dishware) at a table, or a cluster of mugs on a shelf.

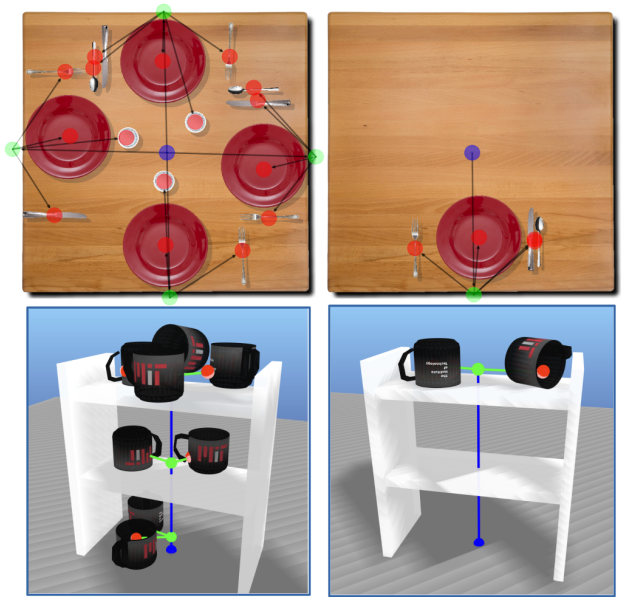

Fig. 1. We generate scenes by sampling scene trees from a generative model that has been conditioned on a training set, and then projecting the resulting object set to the closest physically feasible pose. The original scene trees from before the projection are illustrated here overlaid on the scenes. The distribution of these trees – which controls the number and types of objects and their relative placements – is tuned to match a set of observed environments. The place setting model was trained to match synthetic 2D data, while the mug-on-shelf model was trained to match human-authored 3D data. **Top:** For place settings, the root node of the scene tree is at the center of the table (blue); intermediate table settings can spawn in cardinal directions (green); and place settings can produce a wide variety of cutlery and plates in typical arrangements (red). **Bottom:** For shelves of mugs, the root node is at the bottom of the shelf (blue). Intermediate nodes can spawn on each shelf (green), that themselves spawn mugs in upright, upside-down, and other configurations (orange and red).

When setting up a simulation-based testing environment for a robot, roboticists often create simple, bespoke scripts for generating random testing environments. Even simple scripts can generate remarkably complex distributions of environments – but such off-the-cuff scripting is usually opaque to quantitative analysis. On the other hand, describing

This work is supported by NSF Contract IIS-1427050, an NSF Graduate Research Fellowship under Grant No. 1122374, ONR award N00014-17-1-2699, Lockheed Martin Award No. RPP2016-002, and Lincoln Laboratory/Air Force Award PO# 7000374874.

and doing inference over the distribution of testing environments formally (with e.g. mixture models) is tedious and often limiting. We propose to take the best of both worlds, by using probabilistic programming to maintain the spirit of flexible procedural scripting, while injecting just enough structure to keep inference tractable. Specifically, we propose relying on carefully-specified scene grammars to describe the distribution over the numbers and types of objects, while allowing the continuous distributions over object parameters to be specified flexibly.

II. Related Work

The fields of computer graphics, computer vision, and robotics have, in the past few years, found significant common ground in the problem of "bridging the reality gap." Photorealistic renderers have proven invaluable for providing densely labeled datasets for training deep learning systems for robot perception [1] [2] – with domain randomization [3] and domain adaptation [4] closing the gap even further. Simulators are critical to training data-hungry reinforcement learning systems – with the reality gap likewise usually bridged with domain randomization [5][6], though recent work has looked further more carefully modeling distributions over simulation parameters relating to physics [7] [8].

The particular problem of learning *distributions* over *arrangements of objects* is most closely related to scene understanding and procedural scene generation. Scene trees (and their associated scene grammars) have proven an effective structure for translating between images and natural language for image labeling, retrieval, and generation [9][10][11], and they provide strong structure for understanding point clouds [12] and images [13] [14]. Likewise, they are natural for procedural content generation: [15] uses a scene-tree-like structure to fit a model of complex, highly-structured indoor office environments; and [16] and [17] use a scene grammar as the backbone structure when doing inference over a generative model of human-centric environments. Our framework builds on this previous work on scene grammars: we focus on relaxing the formulation as much as possible while retaining its structure by relying on automated variational inference [18] [19] for the difficult task of model learning. We also focus on scene *parsing* (that is, inferring the scene tree from observations of only the objects) as an essential part of the distribution learning process. Finally, and most generally, we aim probe the usefulness of these techniques for robotics applications.

III. Generative Model

A. Scene Grammars

We define a probabilistic scene grammar $G = (V^T, V^N, R, P)$ with non-terminal symbols classes V^N, terminal symbol classes V^T, production rules R, and rule and production probabilities P. A valid scene tree is rooted at a root symbol of class $V^{root} \in V^N$. Symbol generation follows production rules $R = \{r : A \rightarrow B\}$, where $A \in V^N$ is a symbol class denoting the parent element, and $B \in \{V^N \cup V^T\}$ is a symbol class denoting the child element. (The extension to rules that generate multiple symbols simultaneously is straightforward, but omitted for clarity – see e.g. [20].)

$P = \{p_r, p_v, p_g\}$ describe the probability of generation of a given tree. Each symbol class defines a set of global and local random variables: a symbol class's global variables z_v^g are sampled once and shared across all instances of the class, while its local variables z_v^l are sampled independently for each instance. p_g provides priors for each global variable z_v^g. $p_r(\beta|\alpha)$ describes the probability of generating a symbol instance β (necessarily of class B) from symbol instance α (of class A) using rule r. This generation probability accounts for the dependence of local variables of symbol β on the local and global variables of the parent α – e.g. the relative poses of a child object to its parent – and is assumed to be defined over only continuous variables. Finally, $p_v(r_1, r_2, ..., r_N|v)$ describes the probability, for a symbol instance $v \in V_N$, of the activation of a subset of the legal set of rules $r_i \in R$ that have v as their source. Thus, P is broken into a continuous part (p_g and p_r) describing relationships between symbols, and a discrete part (p_v) describing the distribution over tree structures.

In practice, we further structure our grammar by implementing the AND/OR/SET symbol types popular in the scene parsing literature [13][16]. These symbol types apply additional structure to $p_v(r_1, ..., r_N)$: for example, AND symbols always take all legal rules that start at v, while SET symbols activate each rule as an independent Bernoulli trial. These symbol classes are for convenience and are not an essential or fundamental part of our architecture.

B. Scene Trees

A generated scene tree (also "parse tree" or "scene graph") $t = \{\mathbf{v}, \mathbf{r}\}$ is a tree with a set of symbol instances $\mathbf{v}$ as nodes, with each edge corresponding to the set of active production rules $\mathbf{r}$. (In the extended case where rules can produce multiple symbols, this becomes a hypertree.) A *complete* scene tree is a scene tree in which all nodes and edges have nonzero probability.

A scene tree can be scored by combining the probability of each node and edge. Using $\alpha(r)$ and $\beta(r)$ for the parent and child of rule r, and $\mathbf{r}(v)$ for the children of node v:

$$p(t) = p_g(z^g) \prod_{v \in \mathbf{v}} \Big(p_v(\mathbf{r}(v)|v) \Big) \prod_{r \in \mathbf{r}} \Big(p_r(\beta(r)|\alpha(r)) \Big) \quad (1)$$

where a symbol instance v contributes terms related to its global variables and active child rules, and a rule r contributes a term for its produced child symbols.

C. Concrete example: Table settings

In this example, we use a scene grammar to describe the distribution of typical table settings of dinnerware at a 4-person square table. In this example, all nodes have local random variable representing the planar pose x, y, θ of each object or object group. The set of nodes types is:

1) $V^T = \{Plate, Cup, Fork, Spoon, Knife\}$
2) $V^N = \{Table, PlaceSetting\}$

where the Table node is the only root node.

All generation rules thus define the density of the pose of the generated node given the pose of the parent node. These relative pose distributions are implemented as 3-dimensional diagonal Normal distributions in the parent node frame. Given a target pose offset of a child in parent frame $q_d = \{x_d, y_d, \theta_d\}$, a generation variance Σ_d, and the child pose in the parent frame $q_{rel} = \{x_{rel}, y_{rel}, \theta_{rel}\}$, the density is evaluated as $p_r(\beta|\alpha, q_d, \Sigma_d) = \mathcal{N}_{[\mu=q_d, \Sigma=\Sigma_d]}(q_{rel})$. To support posterior inference of the parameters of these relative pose distributions, q_d and Σ_d are class-specific global random variables with Normal and Inverse-Gamma prior distributions respectively.

The Table has four child production rules for placing a PlaceSetting in each of the four cardinal directions, and the PlaceSetting has production rules for placing each of the terminal node classes within the place setting. The Table and PlaceSetting can both exercise any combination of their rules at any time. A Table has a rule for creating a PlaceSetting at each of the 4 cardinal directions (and hence has $2^4 = 16$ parameters describing the distribution over active rules). A PlaceSetting has a rule for each of 8 dishware placement options: placing a plate, a cup, and placing a fork, knife, or spoon on the left or right of the place setting (and hence has $2^8 = 256$ parameters). The left-right multimodal distribution of cutlery placement is thus captured at the tree structure level. Examples of scenes generated from this grammar are shown in Figures 1 and 4.

We also define a lesioned version of this model to compare performance with the fully-expressive table setting model. This model has node types:

1) $V^T = \{Plate, Cup, Fork, Knife, Spoon\}$
2) $V^N = \{SimpleTable\}$

where the SimpleTable is the only root node, and has the same rule set as a PlaceSetting in the full model. Hence, the model is only expressive enough to describe a single Normal distribution in the Table reference frame for of terminal node type. The SimpleTable has an additional rule to generate another SimpleTable node at the same pose, giving the model the ability to spawn as many of each terminal node type as necessary via recursion.

D. Concrete example: Shelves of Mugs

In this example, we use a scene grammar to describe the distribution of mugs placed neatly onto a three-level shelf. This example captures a practical scenario in which the simulation designer can provide insight that the mugs are going to *tend* to be placed on the shelves in rightside-up or upside-down configurations, but wants to more carefully match the real-world distribution of placements. In this example, all nodes have local random variables representing the 6DOF pose of each object or object group. The set of node types is:

1) $V^T = \{Mug\}$
2) $V^N = \{PreMug, MugShelfLevel, MugShelf\}$

where MugShelf is the only root node. The MugShelf independently chooses to spawn a MugShelfLevel at each of the three shelf levels as Bernoulli trials. Each MugShelfLevel independently chooses whether to spawn each of 6 PreMug instances drawn from the same Normally-distributed pose distribution around the center of the shelf. The parameters of those 6-DOF Normal distributions are drawn themselves from strong priors of the same forms as the planar example. Each PreMug can spawn a single Mug *either* tightly Normally distributed around rightside-up, tightly Normally distributed around upside-down, or broadly (Normally with wide variance) rotationally distributed by choosing from a 3-element Categorical distribution. These final distributions are not parameterized – the purpose of this final node is to provide optional structure for the parser to use to more specifically describe the scenes it observes. Examples of environments generated with this grammar are shown in Figures 1 and 6.

IV. Generation, Inference, and Training

A. Generation

In order to generate a new tree from the grammar, a queue of unexpanded symbol instances is initialized with a root symbol instance, and for each v in the queue that is not a terminal symbol:

- v's global variables z_v^g are sampled from their prior p_g if this is the first node of its class to be expanded.
- The new symbol's child rule set is sampled from p_v.
- For each new child rule r in the rule set, a new symbol is added to the queue by sampling from p_r.

The primary advantage of this procedure is the simplicity and modularity of writing generative models for large scenes. However, some scene-wide constraints relating to physical feasibility – e.g. nonpenetration of all objects – imply some dependence between *all* of the continuous properties of the terminal symbols. While it might be possible to construct a complex set of rules that will only generate physically feasible scenes, we find it more practical to keep the tree generation rules simple and handle feasibility as a postprocessing step. To generate the synthetic training set for the place setting example, we projected generated scenes to the closest nonpenetrating set of object poses via nonlinear optimization, and in the 3D example, then ran a short additional physics simulation to let ensure the scene was statically stable. (An alternative approach would be rejection sampling, but we found the number of samples required to be prohibitively high.)

B. Scene Grammar Parsing

The problem of *scene parsing* is: given a fixed grammar G and a set of observed terminal symbol instances $\mathbf{o} = \{v_1, v_2, ...v_N\}, v_i \in V^T$, sample a scene tree from the posterior distribution of trees $P(t|\mathbf{o})$ that has *exactly* that set of terminal symbol instances. A related simpler problem is to just produce the maximum likelihood tree from that posterior distribution. (The use of the term "parsing" relates to string parsing methods from (e.g. context-free) grammars, as used in natural language processing [21].)

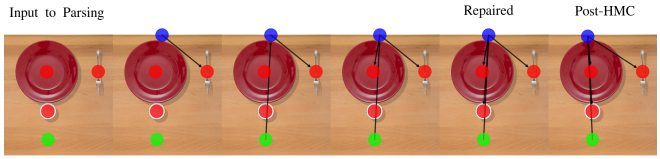

Fig. 2. Demonstration of the scene parsing procedure on a single place setting (a plate, a cup, and a fork) as it incrementally builds a parse tree for a new scene. The input (leftmost) is an incomplete parse tree consisting of only the root node (green) and terminal nodes (red). Each iteration (left to right) adds another intermediate node (blue) to the parse tree. The second-to-final iteration results in complete, repaired tree, and the final iteration (rightmost) performs HMC on the continuous hidden variables to improve the score of the generated tree.

We attempt scene parsing by greedily building an initial feasible tree in a bottom-up fashion starting from the observed terminal symbol instances and a root symbol instance, and then optionally performing some number of MCMC steps over tree configurations by randomly perturbing the tree structure. Our method is very similar in spirit to the beam search technique of [12], and the greedy parsing and simulated annealing of [13]. In practice, we found the initial tree construction step to be by far the most important – for scenes with "obvious" and relatively unambiguous structure, like place settings, the initial tree is very often the optimal tree, and for ambiguous scenes, greedy construction seems empirically likely to arrive at one of the several near-optimal trees. However, we found this method to be far from perfect – see the Discussion for a discussion of when this algorithm does and does not work.

Our scene parsing algorithm uses two subroutines:

- **Tree Completion**: Given an *incomplete* parse tree containing a root symbol instance and a set of parentless nonroot symbol instances, we greedily complete the parse tree by adding new rules and intermediate symbol instances. To accomplish this, for some number of attempts, a random orphan symbol instance v is chosen, and scored against every possible way of giving that instance a parent:
  1) By adding a new production rule to any existing non-terminal symbol instance that has v as its child.
  2) By adding a new symbol instance from v_{new} from V^N, and adding a new production rule that has v_{new} as its source and v as its child.

  Of all of the available above actions, one is selected and applied at random, with higher score actions options taken more often. This procedure is repeated until the tree is feasible.
- **Tree HMC**: Given a complete parse tree, the structure of the tree is fixed while its continuous variables are modified in-place via Hamiltonian Monte Carlo (HMC, [22]) for a small number of iterations. This operation tends to improve the score of the tree, by e.g. shifting the relative poses of objects to higher probability configurations.

The parsing algorithm itself initializes the tree with a root symbol instance and the set of observed terminal symbol instances. This tree is completed using $TreeCompletion$ and optimized with $TreeHMC$ – this first pass greedily samples a (hopefully high-likelihood) tree structure. To recover from incorrect parses, for some number of subsequent passes, a symbol instance is removed from the tree at random, with the symbol being selected inverse proportionally to its score given its children and parent, before the complete-and-HMC procedure is repeated. The resulting tree modification is accepted based on the tree score $p(t)$ (Eq 1) relative to the pre-mutation tree using a Metropolis-Hasting acceptance ratio.

An example of the steps in a parse of a scene is shown in Figure 2. The parsing procedure takes a few seconds per scene in our pure-Python implementation, but is easily parallelizable across data batches.

C. Training Procedure

Given a grammar G parameterized by Θ, and a dataset of (independently chosen) observed terminal node sets $D = [\mathbf{o}_i]$, we seek to find $\arg\max_\Theta \log p_\Theta(D)$. The distribution of our (continuous and discrete) latent variables is indicated as $p(t)$, as it is, in spirit, a distribution over parse trees under our grammar. Applying the standard variational inference methodology, we optimize an approximation to that posterior $q_\Gamma(t)$ parameterized by Γ, and maximize the Evidence Lower Bound Objective (ELBO), which is derived in the standard way (using Jensen's inequality) [18]:

$$\begin{aligned}
\arg\max_\Theta \log p_\Theta(D) &= \log \int_t dt \prod_{\mathbf{o}_i \in D} p_\Theta(\mathbf{o}_i, t) \\
&= \log \int_t dt \prod_{\mathbf{o}_i \in D} p_\Theta(\mathbf{o}_i, t) \frac{q_\Gamma(t)}{q_\Gamma(t)} \\
&= \log \mathrm{E}_{t\sim q_\Gamma(t)}\Big[\prod_{\mathbf{o}_i \in D} \frac{p_\Theta(\mathbf{o}_i, t)}{q_\Gamma(t)}\Big] \\
&\geq \mathrm{E}_{t\sim q_\Gamma(t)}\Big[\log \prod_{\mathbf{o}_i \in D} \frac{p_\Theta(\mathbf{o}_i, t)}{q_\Gamma(t)}\Big],
\end{aligned}$$

leading to our true objective (after converting to a sum over logs, and moving the sum outside of the expectation):

$$\arg\max_{\Theta,\Gamma} \sum_{\mathbf{o}_i \in D} \mathrm{E}_{t\sim q_\Gamma(t)}\Big[\log p_\Theta(\mathbf{o}_i, t) - \log q_\Gamma(t)\Big]. \quad (2)$$

Our choice of $q_\Gamma(t)$ is specialized to our tree model (Eq. 1), and decomposes into separate posteriors over the global and local latent variables. The posterior over the global latent variables (indexing by symbol v) is approximated with mean-field VI using Delta distributions parameterized by Γ $p_g(z^g|D) \approx \prod_{z^g_v} \delta(z^g_v - \Gamma_v)$. Meanwhile, we approximate the posterior over the scene trees (given the global variables) implicitly by sampling the scene trees variables by using the scene parsing method from Section IV-B, and scoring them directly with the relevant part of Eq. 1. This procedure is akin to amortized variational inference, in which data-local latent variables (i.e. the latent parse tree for each data point)

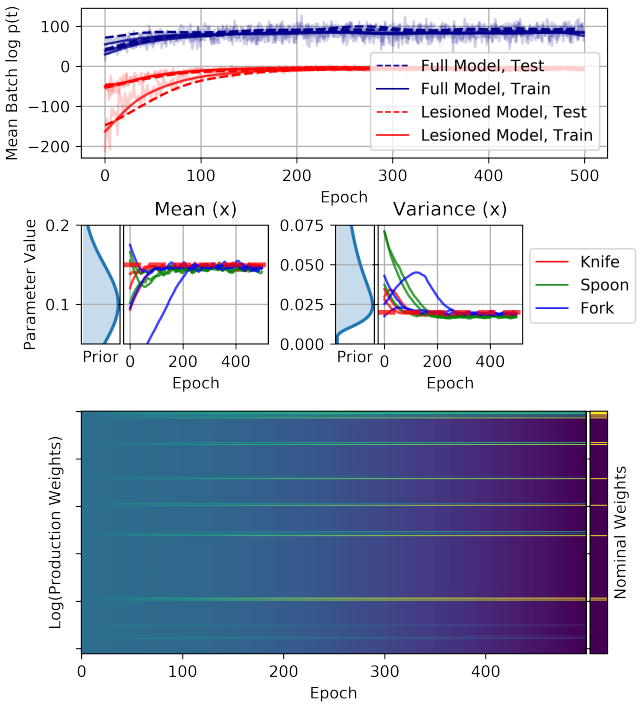

Fig. 3. **Top:** Train and test training curves over epochs for the planar place setting example for 3 runs each of the full model and lesioned model, showing the mean tree score $p(t)$ (Eq 1) across the batch. **Middle:** Trace of estimation of three parameters relating to the left/right (x-axis) position of the right knife, fork, and spoon productions over the 3 training runs for the full model, with the nominal value used to generate the target distribution dashed in red and the prior for the parameters shown on the left. (The right fork, knife, and spoon have the same nominal value and prior in this case.) **Bottom:** A trace across epochs of the $2^8 = 256$ categorical production probabilities for the 8 possible production rules of a PlaceSetting. One vertical slice corresponds to the 256 values at one epoch. At the start of training, the weights are assigned equally (and so all show the same color), but the weights adapt to support the parsed rules over time. The nominal values used to generate the test set are shown on the right with the same color scale.

is sampled using a parameterized function of the data point [23].

This objective is optimized with REINFORCE [24] to estimate the gradient through the expectation: in each epoch, a batch of environments from D are sampled, and for each one, the expectation is approximated with a modest number of samples from $q_\Gamma(t)$. This optimization pushes $q_\Gamma(t)$ closer to the true posterior, as well as optimizing any parameters Θ of the model itself to give more density to observed samples and their corresponding sampled latents (see [18]). We use a moving-average control variate baseline to reduce the variance of the gradient estimate, following [25].

V. Results

All models are implemented in PyTorch [26] and Pyro [19]. We use Adam [27] for choosing variable step sizes for REINFORCE updates – see [19], and specifically [28], for details. For Adam, we use step size 0.01, with a batch size of 10. A single parse tree sample is used to approximate the expectation for each data point. Physical feasibility checks and physics simulations use Drake [29].

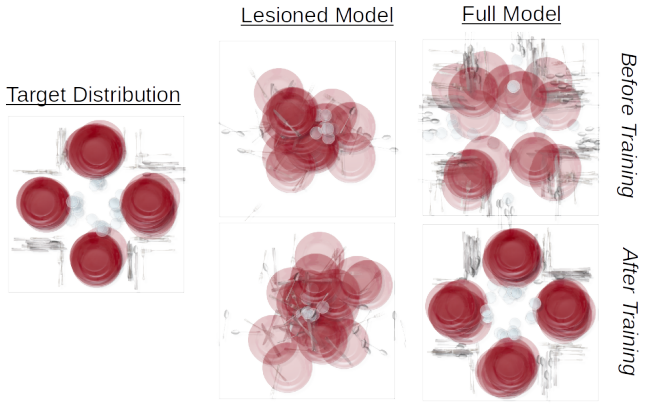

Fig. 4. Each plot shows 20 overlaid samples from a distribution over table settings. **Left:** Samples from the test set. **Middle:** The pre- and post-training distribution of a simplified version of our full model that does not have access to intermediate symbols describing the place settings. **Right:** The pre- and post-training distribution from one training run of our full place setting model.

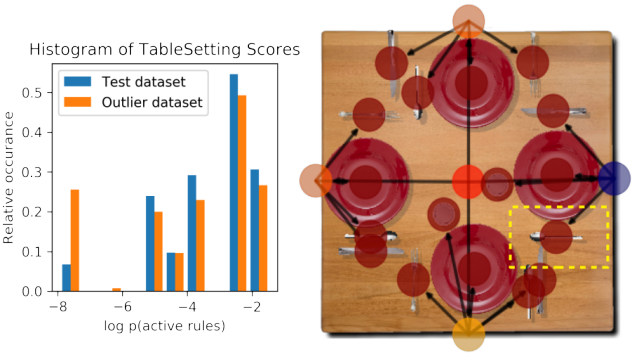

Fig. 5. **Left:** Histogram of the scores of every PlaceSetting that occurs in the maximum-likelihood parse trees for each example in the test set, compared to the scores across a novel dataset that includes left-spoon productions not observed in the training distribution. **Right:** An example of a scene that includes the novel production, with the parse tree nodes color-coded by their score (blue low, red high). The right place setting has abnormally low score because of the left-spoon production (indicated with a yellow box), which never occurs in the training or test set.

A. Planar Place Settings

1) Learning the Distribution: In order to probe the convergence of the inference routine, we created a synthetic dataset of 200 scenes (split into 100 train, 100 test) by manually setting the model parameters and global latent variables to qualitatively-reasonable values and generating samples from the model. We then trained the model from random initializations (drawn from the priors over the latent global variables); loss curves and traces of parameters of interest are shown in Figure 3, and visualizations of the distribution of generated scenes from before and after training are shown in Figure 4. We trained the lesioned model in the same way and include its results in the figures for comparison; as expected, our richer model outperforms the lesioned model.

2) Density Estimation of Novel Scenes: We used the trained model to score all 100 examples from the test set, as well as 100 examples from a different dataset generated

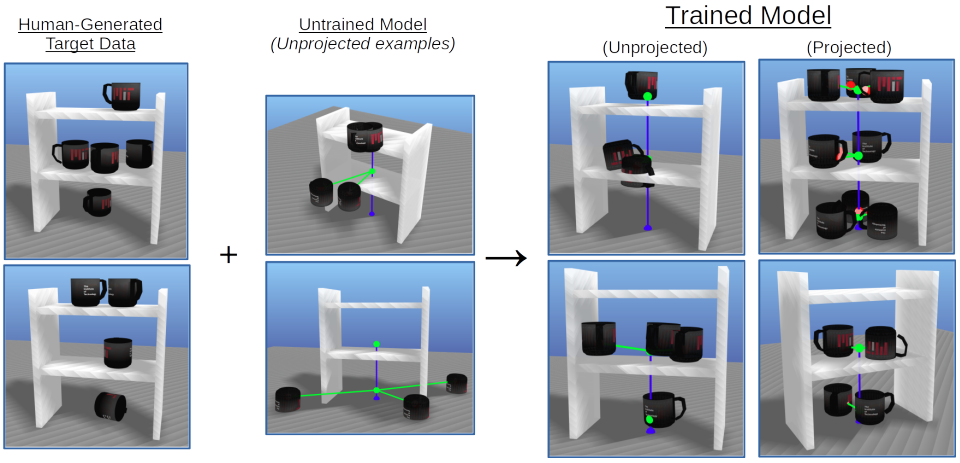

Fig. 6. **Left**: Examples from the human-generated dataset of mug arrangements on a rack. **Middle:** Example scene trees and scenes drawn from the MugShelf model before training, illustrated before being projected to physical feasibility. **Right:** Example scenes drawn from the MugShelf model after training. Scenes from before projection are shown on the left, and from after projection to physically feasible configurations are shown on the right. Even without projection, the model clusters mugs tightly on the shelves, and reliably places them upright and upside down the majority of the time, matching the human distribution. However, post-processing is still required to satisfy the tight constraints of physical feasibility – see Section IV-A.

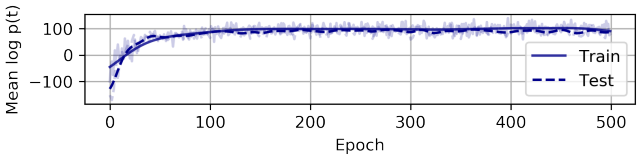

Fig. 7. Train and test training curves over epochs for the MugShelf example, showing the mean tree score $p(t)$ (Eq 1) across the batch.

similarly to the training and test sets, but with an additional generation rule allowing the placement of spoons to the left of the plate. (This placement does not appear in the training or test set.) Results are shown in Figure 5. These results demonstrate that the trained model can be used for relatively sensitive and transparent outlier detection by inspecting the score of each node of the tree for outliers.

B. Shelves of Mugs

To probe whether our method extends to practical 3D scenes that are relevant to robotics, we created a dataset of 100 physically realistic configurations of 1 to 6 mugs on a three-level shelf by motion-controller based teleoperation of a simulation environment. (Using a simulation environment made collecting object poses trivial, while using a motion-controller based interface ensured mug placements were still reasonably natural.) We trained the Shelves-of-Mugs model on this dataset, and report results in Figures 6 and 7.

VI. Discussion and Conclusions

Our scene parsing and variational inference algorithms, in tandem, are capable of learning the distributions of scenes with wildly varying numbers of objects and complex conditional dependence. This style of model has significant advantages: the procedural, modular structure of the grammar made models easy and fast to develop and debug, and the symbol-and-rule-level accounting of the total score of each scene parse was extremely practically useful, even during in development.

The method as presented has plenty of limitations, however. Needing to hand-write the grammar is a major drawback – future work could explore automated creation of new production rules and symbols when the current set is insufficient as a form of model repair. Similarly, requiring that terminal nodes be rigid object instances of fixed class is restrictive; prior work has shown that image features can be used as the terminal nodes in graphs (e.g. [13]), at the cost of greater parsing difficulty. In our experience, the most brittle part of our system is the scene parsing procedure. Greedy parsing needs good guesses of the values of the intermediate instance's variables to succeed – and the correct value may not be obvious, since it may relate to both the instance's parents and children, only some of which are available in the middle of the parsing procedure. An example of a scene where this method did not perform well was in capturing the distribution of dishware in a cluttered dish bin: while there was semantic structure to leverage (e.g. stacks of plates, clusters of the same object), the parsing procedure could not consistently parse the center of clusters, as the center of each cluster is unobserved with an uninformative prior, and the number and identity of its child objects are not obvious.

Practically, this system is at its best fine-tuning models that describe structured scenes that are close to correct but do not quite match reality. In those cases, this model can provide transparent, accurate density estimates for new observed scenes. Scene-grammar-based models have potential for sampling environments for simulation-based testing and for runtime density estimation and outlier detection of task environments. We are particularly excited about the prospect of using structured models like this one as a backbone for simulation-driven rare event and adversarial example search for complex robotic systems (in the style of e.g. [30]).

REFERENCES

[1] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan, "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?" *arXiv preprint arXiv:1610.01983*, 2016.

[2] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," *arXiv preprint arXiv:1809.10790*, 2018.

[3] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[4] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3722–3731.

[5] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[6] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[7] F. Ramos, R. C. Possas, and D. Fox, "Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators," *arXiv preprint arXiv:1906.01728*, 2019.

[8] G. Louppe, J. Hermans, and K. Cranmer, "Adversarial variational optimization of non-differentiable simulators," *arXiv preprint arXiv:1707.07113*, 2017.

[9] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei, "Image retrieval using scene graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3668–3678.

[10] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 129–136.

[11] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1219–1228.

[12] A. Anand and S. Li, "3d scene grammar for parsing rgb-d pointclouds," *arXiv preprint arXiv:1211.1752*, 2012.

[13] Y. Zhao and S.-C. Zhu, "Image parsing with stochastic scene grammar," in *Advances in Neural Information Processing Systems*, 2011, pp. 73–81.

[14] J. Chua and P. F. Felzenszwalb, "Scene grammars, factor graphs, and belief propagation," *arXiv preprint arXiv:1606.01307*, 2016.

[15] M. Fisher, D. Ritchie, M. Savva, T. Funkhouser, and P. Hanrahan, "Example-based synthesis of 3d object arrangements," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, p. 135, 2012.

[16] S. Qi, Y. Zhu, S. Huang, C. Jiang, and S.-C. Zhu, "Human-centric indoor scene synthesis using stochastic grammar," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5899–5908.

[17] E. Jahangiri, R. Vidal, L. Younes, and D. Geman, "Object-level generative models for 3d scene understanding," in *SUNw: Scene Understanding Workshop*, 2014.

[18] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[19] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep Universal Probabilistic Programming," *arXiv preprint arXiv:1810.09538*, 2018.

[20] T. Liu, S. Chaudhuri, V. G. Kim, Q. Huang, N. J. Mitra, and T. Funkhouser, "Creating consistent scene graphs using a probabilistic grammar," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 6, p. 211, 2014.

[21] C. D. Manning, C. D. Manning, and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.

[22] R. M. Neal *et al.*, "Mcmc using hamiltonian dynamics," *Handbook of markov chain monte carlo*, vol. 2, no. 11, p. 2, 2011.

[23] C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt, "Advances in variational inference," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[24] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[25] A. Mnih and K. Gregor, "Neural variational inference and learning in belief networks," *arXiv preprint arXiv:1402.0030*, 2014.

[26] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] "Svi part iii: Elbo gradient estimators." [Online]. Available: https://pyro.ai/examples/svi_part_iii.html

[29] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[30] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," in *Advances in Neural Information Processing Systems*, 2018, pp. 9827–9838.