Learning Object-Based State Estimators for Household Robots Appendix

Yilun Du[1] Tomas Lozano-Perez[1] Leslie Pack Kaelbling[1]

To understand the underlying machinery of OBM-Net, we provide extensive qualitative and quantitative verification of OBM-Net on online clustering in Section I. We further analyze OBM-Net on simple domains with dynamics in Section II and on image domains in Section III. We then provide additional details and results utilizing OBM-Net on simulated household domains in Section IV. We further a derivation of the sparsity loss utilized in training in Section V. Finally, we provide experimental and architecture details for all experiments in both the appendix and main paper using OBM-Net in Section VI.

I. Online Clustering Results

We provide extensive quantitative and qualitative results on online clustering, to illustrate the underlying performance and algorithmic computation performed by OBM-Net. We first compare OBM-Net and baselines across a set of different distributions in Section I-A. We then measure the generalization ability of OBM-Net and baselines in Section I-B. We provide qualitative visualization of OBM-Net in Section I-C, and ablations in Section I-D. Finally, we analyze performance on problems with a larger number of clusters in Section I-E.

A. Evaluation

To check the basic operation of the OBM-Net and to understand the types of problems for which it performs well, we test OBM-Net on online clustering, with data points drawn from different underlying data distributions, each a mixture of three components. We train on 1000 problems drawn from each problem distribution distribution and test on 5000 from the same distribution. In every case, the means of the three components are drawn at random for each problem, with models trained with sequences of length 30.

1) *Normal*: Each component is a 2D Gaussian with fixed identical variance across each individual dimension and across distributions. This is a basic "sanity check."
2) *Elongated*: Each component is a 2D Gaussian, where the variance along each dimension is drawn from a uniform distribution, but fixed across distributions.
3) *Mixed*: Each component is a 2D Gaussian, with fixed identical variance across each individual dimension, but with the variance of each distribution drawn from a uniform distribution.
4) *Angular*: Each component is a 2D Gaussian with identical variance across dimension and distribution, but points above π are wrapped around to $-\pi$ and points below $-\pi$ wrapped to π
5) *Noise*: Each component has 2 dimensions parameterized by Gaussian distributions, but with the values of the remaining 30 dimensions drawn from a uniform distribution centered at 0.

Model	Online	Learned	Normal	Elongated	Mixed	Angular	Noise
OBM-Net	+	+	**0.157**	**0.191**	**0.184**	**0.794**	**0.343**
Set Transformer	+	+	0.407	0.395	0.384	0.794	0.424
LSTM	+	+	0.256	0.272	0.274	0.799	0.408
VQ	+	-	0.173	0.195	0.191	0.992	0.947
Set Transformer	-	+	0.226	0.248	0.274	**0.816**	**0.406**
Slot Attention	-	-	0.254	0.267	0.268	0.823	0.504
K-means++	-	-	**0.103**	**0.139**	**0.135**	0.822	1.259
GMM	-	-	0.113	0.141	0.136	0.865	1.207

TABLE I: **Quantitative Results on Online Clustering.** Comparison of performance on clustering performance across different distributions. Reported error is the L2 distance between predicted and ground truth means. Methods in the bottom half of table operate on observations in bulk and thus are not directly comparable.

In the first three settings, we expect classical clustering methods to perform well, as data is distributed according to the underlying metric space built into classical algorithms. In contrast, the last two settings are challenging for classical clustering methods, as data is not distributed according to the underlying metric space built into the algorithm.

Baselines. To benchmark OBM-Net performance on online clustering, we compare with the following alternative methods: **Batch, non-learning**: K-means++ [1] and expectation maximization (EM) [2] on a Gaussian mixture model (SciKit Learn implementation); **Online, non-learning**: Vector Quantization [3]; **Batch, learning**: Set Transformer [4]; **Online, learning**: LSTM [5], Slot Attention [6] and an online variant of the Set Transformer [4]. All learned network architectures are set to have about 50000 parameters. We provide additional details about architecture and training in Section VI-C and Section VI-B. The Set Transformer is a standard architecture that has been evaluated on clustering problems in the past.

Results. We compare our approach to each of the baselines for the five problem distributions in Table I. The results in this table show that on *Normal*, *Mixed*, and *Elongated* tasks, OBM-Net performs better than learned and constructed online clustering algorithms, but does slightly worse than offline clustering algorithms. Such discrepancy in performance is to be expected due to the fact that OBM-Net is running and being evaluated online. On the *Angular* and *Noise* tasks, OBM-Net is able to learn a useful metric for clustering and outperforms both offline and online alternatives.

B. Generalization

We analyze the ability of OBM-Net to *generalize at test time* to different configurations of input observations,

[1]Computer Science and Artificial Intelligence Laboratory, MIT, USA, yilundu, tlp, lpk@mit.edu.

Model	Online	Learned	Observations			
			10	30	50	100
OBM-Net	+	+	**0.235 (0.001)**	**0.162 (0.001)**	**0.146 (0.001)**	**0.121 (0.001)**
Set Transformer	+	+	0.390 (0.002)	0.388 (0.002)	0.388 (0.002)	0.389 (0.001)
LSTM	+	+	0.288 (0.001)	0.260 (0.001)	0.269 (0.001)	0.288 (0.001)
VQ	+	-	0.246 (0.001)	0.172 (0.001)	0.147 (0.001)	0.122 (0.001)
Set Transformer	-	+	0.295 (0.003)	0.261 (0.001)	0.253 (0.001)	0.247 (0.001)
K-means++	-	-	**0.183 (0.002)**	**0.107 (0.001)**	**0.086 (0.001)**	**0.066 (0.001)**
GMM	-	-	0.189 (0.002)	0.118 (0.001)	0.087 (0.001)	0.067 (0.001)

TABLE II: **Performance with Observation Number.** Comparison of performance after training on one thousand *Normal* distributions for a thousand iterations. We use 3 components, and train models with 30 observations. We report standard error in parentheses. Each cluster observation and center is drawn between -1 and 1, with reported error as the L2 distance between predicted and ground truth mean.

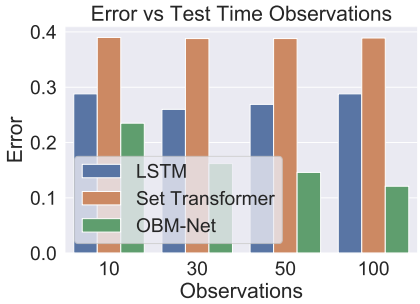

Fig. 1: **Generalization with Increased Observations.** Plot of LSTM, Set Transformer and OBM-Net errors when executed *at test time* on different number of observations from the *Normal* distribution. With increased observations, OBM-Net error continues to decrease while other approaches obtain higher error.

underlying hypothesis slots, and increased number of clusters.

Increased Observation Length. In Table II we evaluate the quality of predictions of OBM-Net and baselines after 10, 30, 50, and 100 observations in the *Normal* distribution at *test time*. We provide additional comparisons across all considered distributions in Table IX. We find that OBM-Net generalizes well to increased numbers of observations, with predictions becoming more accurate as the observation sequence length increases, despite the fact that it is trained only on observation sequences of length 30. This is in contrast to other online learning baselines, set transformer and LSTM, which both see increases in error after 50 or 100 observations. This pattern holds across all other considered test problem distributions.

Increased Hypothesis Slots. In Table III, we investigate the generalization ability of OBM-Net at *test time* to both increases in the number of hypothesis slots and the underlying number of mixture components from which observations are drawn. We compare to the offline set transformer and to VQ, both of which know the correct number of components at test time. Recall that, to evaluate OBM-Net even when it has a large number of extra slots, we use its K most confident hypotheses. We find that OBM-Net generalizes well to increases in hypothesis slots, and exhibits improved performance with large number of underlying components, performing comparably to or better than the VQ algorithm. We note that none of the *learning* baselines can adapt to different numbers cluster components at test time, but find that OBM-Net outperforms the set transformer even when it is trained on the ground truth number of clusters in the test.

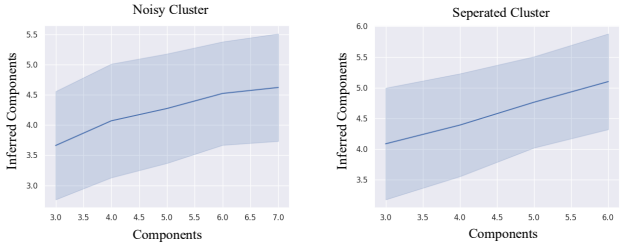

Fig. 2: **Generalization to Increased Cluster Number.** Plots of inferred number of components using a confidence threshold in OBM-Net compared to the ground truth number of clusters (OBM-Net is trained on only 3 clusters). We consider two scenarios, a noisy scenario where cluster centers are randomly drawn from -1 to 1 (left) and a scenario where all added cluster components are well seperated from each other (right). OBM-Net is able to infer more clusters in both scenarios, with better performance when cluster centers are more distinct from each other.

Model	Slots	Ground Truth Clusters		
		3	5	7
OBM-Net	10	**0.162**	0.214	0.242
	20	0.175	**0.195**	0.213
	30	0.188	0.197	**0.205**
Set Transformer	-	0.261	0.279	0.282
Vector Quantization	-	0.171	0.199	**0.205**

TABLE III: **Generalization with Different Hypothesis Slots.** Error of OBM-Net, when executed *at test time* with a different number of hypothesis slots on test distributions with different numbers of ground true components. In all cases, OBM-Net is trained on 3-component problems with 10 slots. OBM-Net achieves good performance with novel number of hypothesis slots, and outperforms different instances of the Set Transformer trained with the ground truth number of cluster components as well as vector quantization.

Inferring Object Number. In contrast to other algorithms, OBM-Net learns to predict both a set of object properties y_k of objects and a set of confidences c_k for each object. This corresponds to the task of both predicting the number of objects in a set of observations, as well as the associated object properties. We further evaluate the ability to regress object number at *test time* in OBM-Net in scenarios where the number of objects is different than that of training. We evaluate on the Normal distribution with a variable number of component distributions, and measure inferred components through a threshold confidence. OBM-Net is trained on a dataset with 3 underlying components. We find in Figure 2 that OBM-Net is able to infer the presence of more component distributions (as they vary from 3 to 6), with improved performance when cluster centers are sharply separated (right figure of Figure 2).

C. Qualitative Visualization

Algorithmic Execution. We provide an illustration of execution of OBM-Net on the *Normal* clustering task in Figure 3 as a trajectory of observations are seen. We plot the decoded values of hypothesis slots in red, with size scaled according to confidence, and ground-truth cluster locations in black. OBM-Net is able to selectively refine slot clusters to be close to ground truth cluster locations even with much longer observation sequences than it was trained on.

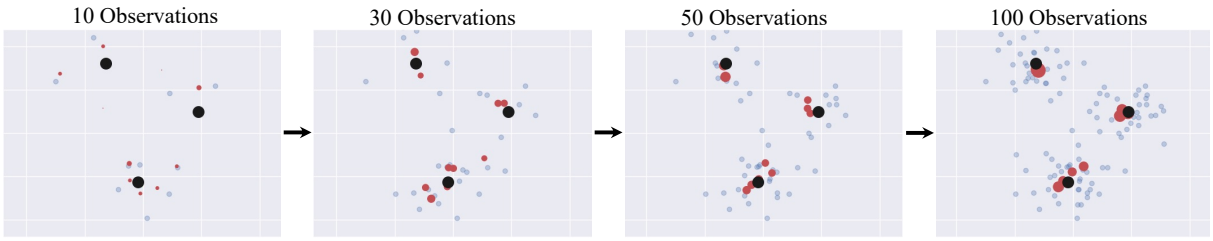

Fig. 3: **Qualitative Visualization of OBM-Net.** Illustration of OBM-Net execution on the *Normal* distribution setting. Decoded value of hypothesis (with size corresponding to confidence) shown in red, with ground truth clusters in black. Observations are shown in blue.

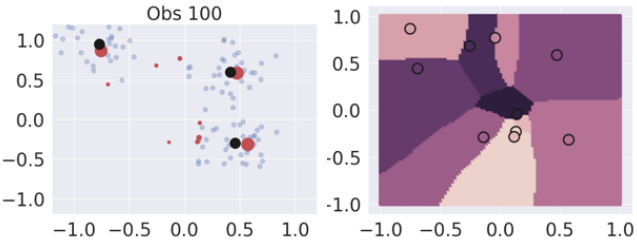

Fig. 4: **Visualization of Attention Weights.** Plot of decoded values of slots (in red) with confidence shown by the size of dot (left), and what slot each input assigns the highest attention towards (right) (each slot is colored differently, with assigned inputs colored in the same way). Note alignment of regions on the right with the decoded value of a slot on the left.

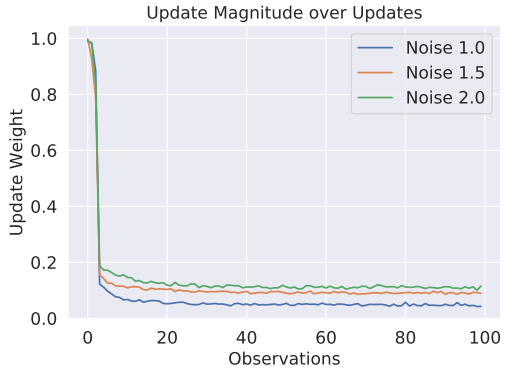

Fig. 5: **Visualization of Relevance Weights.** Plots of the magnitude of relevance weights with increased observation number on different distributions with differing standard deviation (noise).

Submodule Visualization. We find that each component learned by OBM-Net is interpretable. We visualize the attention weights of each hypothesis slot in Figure 4 and find that each hypothesis slot learns to attend to a local region next to the value it decodes to. We further visualize a plot of relevance weights in Figure 5 across an increasing number of observations where individual observations are drawn from levels of noise with respect to cluster centers. We find that as we see more observations, the relevance weight of new observations decreases over time, indicating that OBM-Net learns to pay the most attention towards the first set of observations it sees. In addition, we find that in distributions with higher variance, the relevance weight decreases more slowly, as later observations are now more informative in determining cluster centers.

Sparsity	Learned Memory	Suppression	Relevance	Observations			
				10	30	50	100
-	–	–	–	0.382	0.452	0.474	0.487
+	–	–	–	0.384	0.412	0.423	0.430
+	+	–	–	0.335	0.357	0.366	0.387
+	+	+	–	0.279	0.274	0.278	0.282
+	+	+	+	**0.238**	**0.157**	**0.137**	**0.131**

TABLE IV: **Ablation Analysis.** We ablate each component of OBM-Net on the *Normal* distribution . When learned memory is ablated, OBM-Net updates states based on observed values (appropriate in the *Normal* distribution dataset).

Model	Online	Observations			
		50	65	80	100
OBM-Net	+	**0.158**	**0.154**	**0.151**	**0.147**
VQ	+	0.162	0.157	0.153	0.148
K-means++	-	**0.155**	**0.151**	**0.148**	**0.146**
GMM	-	0.156	0.151	0.149	0.147

TABLE V: **Performance on Large Number of Clusters.** Comparison of performance on *Normal* distribution, when underlying distributions have a large number of components. We use 30 components, and train models with 50 observations. Each cluster observation and center is drawn between -1 and 1, with reported error as the L2 distance between predicted and ground truth means.

D. Ablations

We ablate each component of our model and the results are shown in Table IV and test underlying performance on the *Normal* clustering task. We test removing $\mathcal{L}_{\text{sparse}}$ (sparsity), removing learned slot embeddings (learned memory) — where instead, in individual hypothesis slots, we store the explicit values of inputs, removing the *suppress* modules (supression) and removing the *relevance* module (relevance). We find that each of our proposed components enables better performance on the underlying clustering task. Interestingly, we further find that the addition of *relevance* enables our approach to generalize at test time to larger numbers of observations.

E. Larger Number of Clusters

We measure the performance OBM-Net in the presence of a large number of clusters and slots. We utilize the *Normal* distribution setting, but generate underlying input observations from a total of 30 difference clusters. We train OBM-Net with 50 observations, and measure performance at inferring cluster centers with either 50 or 100 observations. We report performance in Table V and find that OBM-Net approach obtains good performance in this setting, out-performing both online and offline baselines.

Model	Observations			
	10	20	30	40
OBM-Net	**0.415**	0.395	0.382	0.394
Set Transformer	0.699	0.701	0.854	1.007
LSTM	0.422	0.400	0.445	0.464
JPDA (oracle)	0.683	**0.372**	**0.362**	**0.322**

TABLE VI: **Performance on Dynamic Objects.** Comparison of different methods on estimating the state of 3 dynamically moving objects. All learning models are trained with 1000 sequences of 30 observations. We report MSE error. JPDA uses the ground-truth observation and dynamics models.

II. Dynamic Domain Results

We further verify the ability of OBM-Net to perform entity monitoring in a dynamic setting and compare its performance with that of a classical data-association filter.

Setup. We evaluate performance of dynamic entity monitoring using moving 2D objects. A *problem* involves a trajectory of observations z of the K dynamically moving objects, with desired y values being the underlying object positions. Objects evolve under a linear Gaussian dynamics model, with a noisy observation of a single object observed at each step (details in Section VI-A). This task is typical of tracking problems considered by DAF. All learning-based models are trained on observation sequences of length 30. To perform well in this task, a model must discover that it needs to estimate the latent velocity of each object, as well as learn the underlying dynamics and observation models. We utilize $K = 3$ for our experiments.

Baselines. We compare with the de-facto standard method, Joint Probabilistic Data Association (JPDA) [7], which uses hand-built ground-truth models (serving as an oracle). We further compare with our learned online baselines of Set Transformer [4] and LSTM [5] methods.

Result. Quantitative performance, measured in terms of prediction error on true object locations, is reported in Table VI. We can see that the Set Transformer cannot learn a reasonable model at all. The LSTM performs reasonably well for short (length 30) sequences but quickly degrades relative to OBM-Net and JPDA as sequence length increases. We note that OBM-Net performs comparably to, but just slightly worse than, JPDA. *This is strong performance because OBM-Net is generic and can be adapted to new domains given training data without the need to hand-design the models in JPDA*. We believe that these gains are due to the inductive biases built into our architecture.

III. Image Domain Results

We further validate the ability of OBM-Net to perform entity monitoring on image inputs, which requires OBM-Net to synthesize a latent representation for slots, and learn to perform association, update, and transitions in that space.

Setup. We experiment with two separate image-based domain, each consisting of a set of similar entities (2D digits or 3D airplanes). We construct entity monitoring *problems* by selecting K objects in each domain, with the desired y values being images of those objects in a canonical viewpoint. An input observation sequence is generated by randomly selecting one of those K objects, and generating an observation z corresponding to a random viewpoint of the object. Our two domains are: (1) **MNIST**: Each object is a random image in MNIST, with observations corresponding to rotated images, and the desired outputs being the un-rotated images; (2) **Airplane**: Each object is a random object from the Airplane class in ShapeNet [8], with observations corresponding to airplane renderings (using Blender) at different viewpoints and the desired outputs the objects rendered in a canonical viewpoint. We provide details in Section VI-A and use $K = 3$ components.

Model	Learned	MNIST				Airplanes			
Observations		10	30	50	100	10	30	50	100
OBM-Net	+	**7.143**	**5.593**	**5.504**	**5.580**	**4.558**	**4.337**	**4.331**	**4.325**
LSTM	+	9.980	9.208	9.166	9.267	5.106	4.992	4.983	4.998
K-means	+	13.596	12.505	12.261	12.021	7.246	6.943	6.878	6.815

TABLE VII: **Quantitative Results on Image Domain.** Comparison of entity-monitoring performance on MNIST and Airplane datasets across 10, 30, 50, 100 observations. For OBM-Net, LSTM and K-means we use a convolutional encoder/decoder trained on the data. We train models with 30 observations and report MSE error.

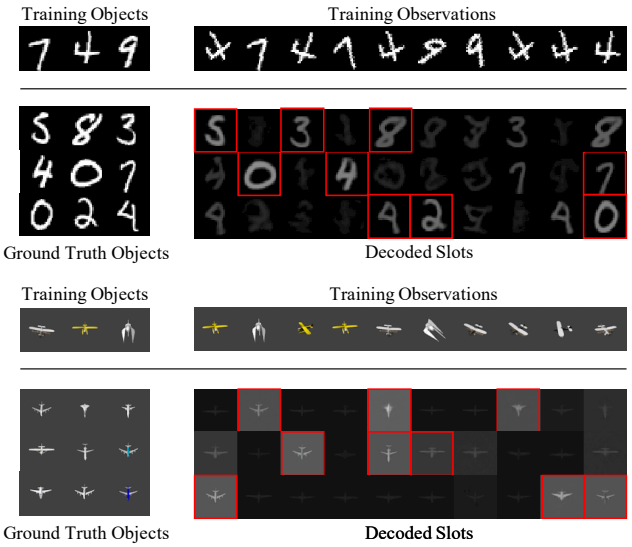

Fig. 6: **Qualitative Visualization of OBM-Net Execution on Images.** Qualitative visualization of two image-based association tasks (left: MNIST, right: airplanes). At the top of each is an example training problem, illustrated by the true objects and an observation sequence. Each of the next rows shows an example test problem, with the ground truth objects and decoded slot values. The three highest-confidence hypotheses for each problem are highlighted in red, and correspond to ground-truth objects.

Baselines. In addition to our learned baselines, we compare with a task specific baseline, batch K-means, in a latent space that is learned by training an autoencoder on the observations. In this setting, we were unable to train the Set Transformer stably and do not report results for it.

Results. In Table VII, we find that our approach significantly outperforms other comparable baselines in both accuracy and generalization. We further visualize qualitative predictions from our model in Figure 6. We find that our highest confidence decoded slots correspond to ground truth objects.

Model	Learned	Table Accuracy			Position Error		
Observations		10	25	50	10	25	50
OBM-Net	+	**0.992**	**0.925**	**0.813**	**0.159**	**0.234**	**0.301**
LSTM	+	0.883	0.625	0.489	0.203	0.294	0.354
Clustering	-	0.798	0.638	0.554	0.204	0.266	0.318

TABLE VIII: **Quantitative Performance on Simulated iGibson Houses.** Comparison of performance of OBM-Net and baselines on a iGibson house.

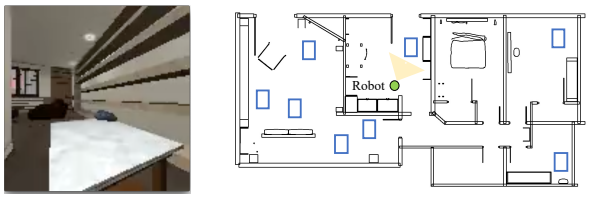

Fig. 7: **Visualization of Simulated iGibson Environment.** (left) Illustration of example RGB input in our iGibson environment. (right) Example configuration of tables in our iGibson environment (tables drawn in blue).

IV. Household Domains Results

To further validate the efficacy of OBM-Net on a simulated household robotic domains, we test OBM-Net on an interactive house found in the iGibson environment. We illustrate an example input observation of our environment, and the corresponding configuration of tables and objects in Figure 7. We utilize the same configuration settings as the Pybullet environment, training models on trajectories of length 50, consisting of 8 tables with 2 objects on them each. We utilize object classes and movement patterns from Configuration A described in Section 5.4 in the main paper. We provide additional dataset details in Section VI-A.

We compare OBM-Net with LSTM and clustering baselines discussed in Section 5.4 of the main paper. We use the same metrics as described in Section 5.4. In Table VIII we report results on this household setting. We find that OBM-Net performs significantly better than LSTM and Clustering baselines.

V. Sparsity Loss

In this section, we show that $\mathcal{L}_{\text{sparse}}(\mathbf{c})$ encourage confidences $\mathbf{c}$ to be sparse. Recall that

$$\mathcal{L}_{\text{sparse}}(\mathbf{c}) = -\log\|\mathbf{c}\| \quad . \tag{1}$$

where $\|c\|$ is the L2 norm which is convex. Recall that c, the confidence vector, defines a polyhedron, since it is the set of points that are non-negative, and whose elements sum up to one. The maximum of a convex function over a polyhedra must occur at the vertices, which correspond to an assignment of 1 to a single c_i and 0s to every other value of $\mathbf{c}$. Next we consider the minimum of $\|c\|$ given that its elements sum up to one. This is equivalent to finding the stationary points of the Legragian

$$\sum_i c_i^2 + \lambda(\sum_i c_i - 1) \tag{2}$$

Fig. 8: **Qualitative Visualization of Domains.** Visualizations of the Normal Gaussian, Dynamic domains and Simulated Household domains. Observations are transparent while ground truth states are bolded for gaussian and dynamic domains. Four sample image observations shown for the simulated household robotic domain.

By taking the gradient of the above expression, we find that the stationary value corresponds to each c_i being equal. Since the function is convex, this corresponds to the minimum of $\|c\|$. Thus $\mathcal{L}_{\text{sparse}}(\mathbf{c})$ is maximized when each individual confidence is equal.

VI. Experimental Details

In this section, we provide details of our experimental approach. We first discuss the details of datasets used in Section VI-A. Next, we provide the model architectures used in Section VI-B. Finally, we provide details on the baselines we compare with in Section VI-C.

A. Dataset Details

We first provide detailed experimental settings for each of the datasets considered in the paper.

Online Clustering. In online clustering, we utilize observations drawn from the following distributions, where Gaussian centers are drawn uniformly from -1 to 1.

1) *Normal*: Each 2D Gaussian has standard deviation 0.2. The normal setting is illustrated in Figure 8.
2) *Mixed*: Each distribution is a 2D Gaussian, with fixed identical variance across each individual dimension, but with the standard deviation of each distribution drawn from a uniform distribution from (0.04, 0.4).
3) *Elongated*: Each distribution is a 2D Gaussian, where the standard deviation along each dimension is drawn from a uniform distribution from (0.04, 0.4), but fixed across distributions.
4) *Angular*: Each distribution is a 2D Gaussian with identical standard deviation across dimension and distribution, but points above π are wrapped around to $-\pi$ and points below $-\pi$ wrapped to π. Gaussian means are selected between $(-\pi, -2\pi/3)$ and between $(2\pi/3, \pi)$. The standard deviation of distributions is $0.3 * \pi$.
5) *Noise*: Each distribution has 2 dimensions parameterized by Gaussian distributions with standard deviation 0.5, but with the values of the remaining 30 dimensions drawn from a uniform distribution from $(-1, 1)$.

Dynamic Domains. Next, in the dynamics domain, we implement our dataset using the StoneSoup library*. We initialize the location of each cluster from a Gaussian distribution with standard deviation 1.5 and initialize velocity in each directory from a Gaussian distribution with standard deviation of 0.02. At each timestep, Gaussian noise is added to velocities with magnitude 1e-4. We show example tracks

*https://stonesoup.readthedocs.io/en/v0.1b3/stonesoup.html

in Figure 8. Our JPDA implementation is also from the StoneSoup library.

Image Domains. In the image domain, for MNIST, we use the 50000 images in the training set to construct the training problems, and the 10000 images in the non-overlaping test set to construct the test problems. For the Airplane dataset, we use 1895 airplanes to construct the training problems, and 211 different airplanes to construct the test problems. Each airplane is rendered with 300 viewpoints.

Simulated Household Domains. For the simulated household robotics domains, we implement our embodied house environment Pybullet, and construct a house with x and y axis between -1 and 1. We utilize furniture assets from [9] to each of the individual classes of objects considered, with 50% of the objects (sorted alphabetically) being used for the training dataset and the 50% of the objects used for the test dataset. Each individual object is scaled by a factor of 0.1 to fit on each table. Each table has size 0.15 by 0.1 in our setting. Our constructed house environment uses the floor plan illustrated in Figure 1 of the main paper. Across configurations, objects, which each individual step of the trajectory corresponding to a $1/60$ of second advancement of simulation time in PyBullet. In each configuration, objects in separate object classes move with velocities of $(0.6, 0.0)$, $(0.0, 0.6)$ and $(0.3, 0.3)$ units per second respectively. In the presence of collision, all objects involved in the collision event stop moving, making the underlying dynamics of objects a stochastic process.

For the iGibson house results included in the appendix, we utilize the Pomaria environment in iGibson environment. This house has x and y axis roughly between -4 and 4, and thus we scale tables to a size of 0.45 by 0.3 in the environment, and proportionally scale up the size of individual objects as well as their underlying movement speed. To sample trajectories in both settings, we sample a set of points across rooms in a house and utilize motion planning to infer paths connecting each individual point.

B. Model/Baseline Architectures

We provide the overall architecture details for the LSTM in Figure 9a, for the set-transformer in Figure 9b and OBM-Net in Figure 10a. For image experiments, we provide the architecture of the encoder in Figure 11 and decoder in Figure 12. Both LSTM, OBM-Net, and autoencoding baselines use the same image encoder and decoder. For robotics experiments, we provide the architecture of the shape decoder in Figure 10b.

In OBM-Net memory, the function update(s_k, n_k, e) is implemented by applying a 2 layer MLP with hidden units h which concatenates the vectors s_k, n_k, e as input and outputs a new state u_k of dimension h. The value n_k is encoded using the function $\frac{1}{1+n_k}$, to normalize the range of input to be between 0 and 1. The function attend(s_k, n_k, e) is implemented in an analogous way to update, using a 2 layer MLP that outputs a single real value for each hypothesis slot.

For the function relevance(s_k, n_k, e), we apply NN_1 per hypothesis slot, which is implemented as a 2 layer MLP with hidden units h that outputs a intermediate state of dimension h. (s_k, n_k, e) are fed into NN_1 in an analogous manner to update. NN_2 is applied to average of the intermediate representations of each hypothesis slot and is also implemented as a 2 layer MLP with hidden unit size h, followed by a sigmoid activation. We use the ReLU activation for all MLPs. NN_3 is represented is GRU, which operates on the previous slot value.

C. Baseline Details

All baseline models are trained using prediction slots equal to the ground truth of components. To modify the set transformer to act in an online manner, we follow the approach in [10] and we apply the Set Transformer sequentially on the concatenation of an input observation with hypothesis slots. Hypothesis slots are updated based off the new values of the slots after applying self-attention (Set Transformer Encoder). We use the Chamfer loss to train baseline models, with confidence set to 1.

Dense $\rightarrow$ h
Dense $\rightarrow$ h
LSTM(h)
Dense $\rightarrow$ h
Dense $\rightarrow$ output

(a) The model architecture of the LSTM baseline. The hidden dimension h used is 96 for synthetic Gaussian distributions and 128 for Image datasets. For image experiments, the first 2 and last 2 fully connected layers are replaced with image encoders and decoders.

Dense $\rightarrow$ h
Dense $\rightarrow$ h
Set Transformer Encoder
Set Transformer Decoder

(b) The model architecture of the Set Transformer baseline. The hidden dimension h is 48 for the synthetic Gaussian distributions. We use the encoder and decoder defined in [4] with 4 heads and hidden dimension h.

Fig. 9: Architecture of baseline models.

Dense $\rightarrow$ h
Dense $\rightarrow$ h
OBM-Net Memory
Dense $\rightarrow$ h
Dense $\rightarrow$ output

(a) The model architecture of OBM-Net. The hidden dimension h is 64 is for synthetic Gaussian distributions and 128 for the image/robotics experiments. For image experiments, the first and last 2 linear layers are replaced with convolutional encoders and decoders.

$(x, y, z) \rightarrow$ Dense $\rightarrow$ h
Concat$(h,$ state$)$
Dense $\rightarrow$ h
Dense $\rightarrow$ 1

(b) The shape decoder of OBM-Net used in the robotics experiments. The shape decoder takes as input a voxel coordinate as well as a slot value and predicts a occupancy for the voxel.

Fig. 10: Architecture of OBM-Net and the shape decoder.

5x5 Conv2d, 32, stride 2, padding 2
3x3 Conv2d, 64, stride 2, padding 1
3x3 Conv2d, 64, stride 2, padding 1
3x3 Conv2d, 64, stride 2, padding 1
3x3 Conv2d, 128, stride 2, padding 1
Flatten
Dense $\rightarrow$ h

Fig. 11: The model architecture of the convolutional encoder for image experiments.

Dense $\rightarrow$ 4096
Reshape $(256, 4, 4)$
4x4 Conv2dTranspose, 128, stride 2, padding 1
4x4 Conv2dTranspose, 64, stride 2, padding 1
4x4 Conv2dTranspose, 64, stride 2, padding 1
4x4 Conv2dTranspose, 64, stride 2, padding 1
3x3 Conv2d, 3, stride 1, padding 1

Fig. 12: The model architecture of the convolutional decoder for image experiments.

Type	Model	Online	Observations			
			10	30	50	100
Normal	OBM-Net	+	0.235	0.162	0.146	0.128
	Set Transformer	+	0.390	0.388	0.388	0.389
	LSTM	+	0.288	0.260	0.269	0.288
	VQ	+	0.246	0.172	0.147	0.122
	Set Transformer	+	0.295	0.261	0.253	0.247
	K-means++	-	0.183	0.107	0.086	0.066
	GMM	-	0.189	0.118	0.087	0.067
Mixed	OBM-Net	+	0.255	0.184	0.164	0.147
	LSTM	+	0.306	0.274	0.284	0.290
	Set Transformer	+	0.415	0.405	0.407	0.408
	VQ	+	0.262	0.192	0.169	0.145
	Set Transformer	-	0.309	0.274	0.266	0.261
	K-means++	-	0.206	0.135	0.105	0.088
	GMM	-	0.212	0.136	0.105	0.079
Enlongated	OBM-Net	+	0.258	0.192	0.173	0.161
	LSTM	+	0.314	0.274	0.288	0.300
	Set Transformer	+	0.394	0.391	0.394	0.394
	VQ	+	0.265	0.194	0.172	0.149
	Set Transformer	-	0.309	0.244	0.240	0.232
	K-means++	-	0.213	0.139	0.113	0.092
	GMM	-	0.214	0.141	0.112	0.086
Rotation	OBM-Net	+	0.892	0.794	0.749	0.736
	LSTM	+	0.799	0.796	0.795	0.794
	Set Transformer	+	0.793	0.794	0.782	0.782
	VQ	+	0.956	1.000	1.000	0.984
	Set Transformer	-	0.815	0.784	0.779	0.772
	K-means++	-	0.827	0.834	0.823	0.802
	GMM	-	0.842	0.875	0.867	0.848
Noise	OBM-Net	+	0.375	0.343	0.338	0.334
	LSTM	+	0.419	0.406	0.405	0.407
	Set Transformer	+	0.434	0.424	0.425	0.424
	VQ	+	1.479	0.948	0.826	0.720
	Set Transformer	-	0.436	0.407	0.398	0.394
	K-means++	-	1.836	1.271	1.091	0.913
	GMM	-	1.731	1.215	1.056	0.856

TABLE IX: **Generalization with Increased Observations.** Error of different models when executed *at test time* on different number of observations across different distributions. We train models with 3 components and 30 observations.

REFERENCES

[1] David Arthur and Sergei Vassilvitskii. "k-means++: the advantages of careful seeding". In: *Symposium on Discrete Algorithms '07*. 2007.

[2] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[3] R. Gray. "Vector quantization". In: *IEEE ASSP Magazine* 1.2 (1984), pp. 4–29.

[4] Juho Lee et al. "Set transformer: A framework for attention-based permutation-invariant neural networks". In: *arXiv preprint arXiv:1810.00825* (2018).

[5] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Comput.* 9.8 (1997), pp. 1735–1780.

[6] Francesco Locatello et al. *Object-Centric Learning with Slot Attention*. 2020. arXiv: 2006.15055 [cs.LG].

[7] Yaakov Bar-Shalom, Fred Daum, and Jim Huang. "The Probabilistic Data Association Filter". In: *IEEE Control Systems Magazine* (2009).

[8] Angel X Chang et al. "Shapenet: An information-rich 3d model repository". In: *arXiv:1512.03012* (2015).

[9] Fei Xia et al. "Interactive Gibson Benchmark: A Benchmark for Interactive Navigation in Cluttered Environments". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 713–720.

[10] Adam Santoro et al. "Relational recurrent neural networks". In: *Advances in neural information processing systems*. 2018, pp. 7299–7310.