

Fine-Tuning Generative Models as an Inference Method for Robotic Tasks

Anonymous Author(s)

Affiliation

Address

email

1 **Abstract:** Adaptable models could greatly benefit robotic agents operating in the
2 real world, allowing them to deal with novel and varying conditions. While ap-
3 proaches such as Bayesian inference are well-studied frameworks for adapting
4 models to evidence, we build on recent advances in deep generative models which
5 have greatly affected many areas of robotics. Harnessing modern GPU accelera-
6 tion, we investigate how to quickly adapt the sample generation of neural network
7 models to observations in robotic tasks. We propose a simple and general method
8 that is applicable to various deep generative models and robotic environments.
9 The key idea is to quickly fine-tune the model by fitting it to generated samples
10 matching the observed evidence, using the cross-entropy method. We show that
11 our method can be applied to both autoregressive models and variational autoen-
12 coders, and demonstrate its usability in object shape inference from grasping, in-
13 verse kinematics calculation, and point cloud completion.

14 1 Introduction

15 Humans and other animals maintain powerful mental models of the world [1] for navigation, object
16 manipulation, social interaction and other day-to-day tasks. These mental models are imperfect and
17 tend to be inaccurate. However, they are highly adaptable and are frequently updated upon arrival
18 of new information. Robotic agents operating in diverse and unstructured environments may also be
19 required to adjust their behavior, and would therefore benefit from adaptable models similar to the
20 ones humans hold. Recent studies in robot manipulation and navigation have focused on scenarios
21 where an accurate model can be learned and used as-is in downstream planning tasks [2, 3, 4].
22 In contrast, inspired by biological mental models, this paper focuses on how to efficiently adapt a
23 model to novel information.

24 Natural approaches to updating models given new evidence such as *Bayesian inference* have been
25 used extensively in computer vision and robotics [5, 6]. When the modelled probability distributions
26 are low-dimensional, Bayesian inference may have closed form solutions or the posterior can be nu-
27 merically estimated using methods such as Markov chain Monte Carlo (MCMC, [7]). However,
28 there is growing interest in using high-dimensional *deep generative models* for robotics, which can
29 represent complex and diverse data. Advances such as *variational autoencoders* [8] and *diffusion*
30 *models* [9] make it possible to learn diverse and high-dimensional distributions. For such expressive
31 models, Bayesian inference techniques are not suitable, and they cannot operate on the time-scale
32 required for robotic tasks [10]. Another approach, which has been found effective with deep gener-
33 ative models, is to *train* them to adapt by conditioning on possible observations [11, 12]. However,
34 this paradigm may fall short when faced with out-of-distribution evidence at test-time.

35 We propose a simple approach for updating the parameters of deep generative models given empiri-
36 cal observations, to approximate complex posterior distributions. Our method requires a forward
37 simulation of the environment which can produce observations given a model, and a similarity func-
38 tion for observations. We build on GPU-based physics simulation [13] and model training to perform

39 fast inference in a novel robotic scenario by fine-tuning the generative model weights. Consider-
40 ing the parametric nature of deep generative models, we use a version of the cross-entropy method
41 (CEM [14]) to quickly update the parameters to generate observations conforming with the available
42 evidence. We dub our method MACE, for **Model Adaptation with the Cross-Entropy** method.

43 To showcase MACE, we focus on robotic domains where the forward simulation step is easily per-
44 formed using off-the-shelf physical simulators. We demonstrate the versatility of MACE on sev-
45 eral robotic manipulation tasks that we frame as model adaptation, using two different types of
46 deep generative models. In particular, we demonstrate results on object identification from position
47 measurements of a multi-fingered robot gripper, on recovery of object point clouds given partial
48 measurements (as generated by depth sensors) and on an inverse kinematics (IK) task in the pres-
49 ence of obstacles. In all of these environments, the posterior has a rich multi-modal structure. We
50 demonstrate that MACE is indeed capable of producing diverse posterior samples for a variety of
51 observations, and that it outperforms baseline approaches in diversity and accuracy. In terms of
52 speed, we show that by exploiting GPU-based simulation and inference, our fine-tuning can be pre-
53 formed online in a competitive time frame. For example, in the IK tasks, we find that MACE can
54 outperform the MoveIt [15] library in quickly finding IK solutions for complex scenarios.

55 **2 Related Work**

56 *Bayesian inference* is concerned with computing the posterior distribution of the model given obser-
57 vations. Exact computation is possible for simple models that admit conjugate priors, and Markov
58 chain Monte Carlo (MCMC) can be used for general models [7]. Approximate Bayesian compu-
59 tation (ABC [16]) allows sampling from the posterior without an exact likelihood, but with some
60 similarity function between observations. MACE is inspired by these approaches, and expedites the
61 search for better samples using the cross-entropy method (CEM [14]). Recent work by Engel et al.
62 [17] introduces a Bayesian model update scheme using CEM which is related to ours. However,
63 Engel et al. [17] require the true likelihood, and their method is limited to relatively small models,
64 while ours can be applied to deep generative models.

65 Bayesian inference has been used extensively in robotics in the context of state estimation, localiza-
66 tion, and mapping [5, 18]. In recent work, Marlier et al. [19] use Bayesian estimation of a posterior
67 distribution of grasp poses for multi-finger object grasping, and Pastor et al. [10] use Bayesian
68 inference with LSTM [20] to classify objects using tactile sensors. Both the above are application-
69 specific, while MACE is a general approach applicable to a variety tasks and generative models.

70 Another approach is to amortize inference by learning an approximate posterior using data from
71 the joint distribution $p(\mathbf{x}, \mathbf{o})$ such as in the *conditional variational autoencoder* (CVAE) model
72 [21]. We compare MACE with a CVAE baseline, and show that while our inference procedure is
73 slower, MACE produces a more diverse posterior. Furthermore, since MACE leverages a similarity
74 function between observations, it can work for observations that are out of distribution with respect
75 to $p(\mathbf{x}, \mathbf{o})$, unlike the CVAE. Finally, MACE can tune the same prior model with different modalities
76 of observations without retraining.

77 *Meta-learning*, and *meta-RL* in particular, is an alternative approach to quickly adapt behavior to
78 new evidence [22]. However, meta-RL is typically model-free, and learns how to adapt a policy [22,
79 23]. Model-based meta-RL approaches such as Zintgraf et al. [12] use a CVAE to condition on the
80 history of observations. Whether such methods could be improved using our inference method is an
81 interesting direction for future research.

82 **3 Model Adaptation with the Cross-Entropy Method**

83 We now describe MACE, our method for adapting deep generative models to environment observa-
84 tions using the cross-entropy method. We begin by describing the setup and the types of tasks we
85 aim to solve; next, we discuss our update rule and present the full algorithm.

86 **3.1 Problem Formulation**

87 In the robotics context, an *inference problem* involves the recovery of *task* parameters given obser-
 88 vations of the environment. We assume some distribution $p(\mathbf{x})$ over task representations $\mathbf{x} \in \mathcal{X}$.
 89 A task description \mathbf{x} could be a point cloud (PC) of an object as in a grasping task; an image input
 90 for more complex manipulation; or a desired joint configuration in a reaching task. MACE requires
 91 access to a generative model representing a parametric distribution over task representations $p(\mathbf{x}; \theta)$.
 92 We assume the generative model is initially trained to represent the prior, i.e., for some initial pa-
 93 rameter θ_0 it holds that $p(\mathbf{x}; \theta_0) = p(\mathbf{x})$.

94 Although the task representation \mathbf{x} is unknown, some information about it can be observed. We
 95 denote this information $\mathbf{o} \in \mathcal{O}$ (for *observation*) and note that it can be of any form, For example,
 96 in a grasping task, \mathbf{o} could be a partial PC obtained from a depth sensor (while \mathbf{x} is the full object
 97 model); in a reaching task, \mathbf{o} could represent obstacles for the robot to avoid.

98 A central component of MACE is a *simulator* of the task. Exact environment simulators are often
 99 hard to design, leaving gaps to the reality they attempt to simulate [18]. However, for our purposes,
 100 we only require the simulator to emit observations of a similar modality to the ones emitted by the
 101 environment. Therefore, the simulator can be viewed as a probability distribution $\hat{p}(\mathbf{o}|\mathbf{x})$, providing
 102 observations given task representations.

103 The goal of the inference task is to train the parametric model $p(\mathbf{x}; \theta)$ to produce a distribution
 104 closely resembling the posterior $p(\mathbf{x}|\mathbf{o})$.

105 **3.2 Updating the Model**

106 We aim to update the model parameters θ so that the generative model $p(\mathbf{x}; \theta)$ more closely re-
 107 sembles the posterior $p(\mathbf{x}|\mathbf{o})$. We can do this by minimizing the Kullback-Leibler (KL) divergence
 108 between the posterior and the parametric model:

$$\arg \min_{\theta} D_{\text{KL}}(p(\mathbf{x}|\mathbf{o})||p(\mathbf{x}; \theta)) = \arg \min_{\theta} \int p(\mathbf{x}|\mathbf{o}) \log p(\mathbf{x}|\mathbf{o}) d\mathbf{x} - \int p(\mathbf{x}|\mathbf{o}) \log p(\mathbf{x}; \theta) d\mathbf{x}.$$

109 The parametric model is only present in the second term, therefore we can maximize it
 110 to minimize the entire KL divergence. Since the posterior $p(\mathbf{x}|\mathbf{o})$ is unknown, we use
 111 Bayes' rule to replace it with the likelihood $p(\mathbf{o}|\mathbf{x})$: $\arg \max_{\theta} \int p(\mathbf{x}|\mathbf{o}) \log p(\mathbf{x}; \theta) d\mathbf{x} =$
 112 $\arg \max_{\theta} \int \frac{p(\mathbf{o}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{o})} \log p(\mathbf{x}; \theta) d\mathbf{x} = \arg \max_{\theta} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} [p(\mathbf{o}|\mathbf{x}) \log p(\mathbf{x}; \theta)]$.

113 The likelihood of the observation $p(\mathbf{o}|\mathbf{x})$ is also an unknown quantity. However, we may estimate it
 114 using the forward simulator, which can produce observations \mathbf{o} given \mathbf{x} . We define a *score function*
 115 as any function $S : \mathcal{O} \times \mathcal{O} \rightarrow [0, 1]$ indicating similarity between pairs of observations, and assume
 116 $\mathbb{E}_{\mathbf{o}' \sim \hat{p}(\mathbf{o}|\mathbf{x})} S(\mathbf{o}', \mathbf{o})$ is an estimate of $p(\mathbf{o}|\mathbf{x})$. An intuitive case to justify this assumption is when
 117 observations \mathbf{o} are discrete, and $S(\mathbf{o}', \mathbf{o}) = \mathbf{1}_{\mathbf{o}'=\mathbf{o}}$ is an indicator of whether \mathbf{o}' is equal to the
 118 evidence \mathbf{o} ¹. In practice, as our simulators are deterministic, we replace the expectation with a single
 119 observation \mathbf{o}' per sample \mathbf{x} . In the following, we abuse notation by referring to $\mathbb{E}_{\mathbf{o}' \sim \hat{p}(\mathbf{o}|\mathbf{x})} S(\mathbf{o}', \mathbf{o})$
 120 as $S(\mathbf{o}', \mathbf{o})$. Plugging in the score function, the optimization problem becomes:

$$\arg \max_{\theta} \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} [S(\mathbf{o}', \mathbf{o}) \log p(\mathbf{x}; \theta)]. \quad (1)$$

121 Recalling our assumption that $p(\mathbf{x}; \theta_0) = p(\mathbf{x})$, Eq. 1 can be optimized using importance sam-
 122 pling: $\arg \max_{\theta} \frac{1}{N} \sum_{i=1}^N \frac{p(\mathbf{x}_i; \theta_0)}{p(\mathbf{x}_i; \theta)} S(\mathbf{o}_i, \mathbf{o}) \log p(\mathbf{x}_i; \theta)$, where $\mathbf{x}_i \sim p(\mathbf{x}; \theta)$, the sampling distri-
 123 bution. One question, however, is how to choose an effective sampling distribution which places
 124 enough mass on high-scoring \mathbf{x} values. Inspired by the iterative approach of Engel et al. [17], we
 125 optimize this objective iteratively using stochastic gradient descent. At each iteration, we use the
 126 parametric model from the previous iteration as the sampling distribution and take a few gradient

¹For other examples of score functions, see the environment descriptions in Sec. 4.

127 steps to obtain the next model parameters. The objective at iteration t is given by:

$$\arg \max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^N \frac{p(\mathbf{x}; \boldsymbol{\theta}_0)}{p(\mathbf{x}; \boldsymbol{\theta}_{t-1})} S(\mathbf{o}_i, \mathbf{o}) \log p(\mathbf{x}_i; \boldsymbol{\theta}), \quad (2)$$

128 with $x_i \sim p(\mathbf{x}; \boldsymbol{\theta}_{t-1})$. In practice, we find that the importance sampling term in this objective makes
 129 it difficult to optimize due to large discrepancies between the parametric distributions². Instead, we
 130 introduce another approximation and remove the importance sampling term $\frac{p(\mathbf{x})}{p(\mathbf{x}; \boldsymbol{\theta})}$ from Eq. 2.

131 To further improve performance and
 132 shorten training times, we follow the
 133 cross-entropy method (CEM) formu-
 134 lation described by Botev et al. [24].
 135 We view the objective in Eq. 1 as
 136 the problem of finding a distribu-
 137 tion $p(\mathbf{x}; \boldsymbol{\theta})$ which produces samples
 138 with high scores $S(\mathbf{o}', \mathbf{o})$. To op-
 139 timize $S(\mathbf{o}', \mathbf{o})$ (which is implicitly
 140 a function of \mathbf{x} through the simula-
 141 tor), we treat $p(\mathbf{x}; \boldsymbol{\theta})$ as an impor-
 142 tance sampling distribution and ad-
 143 just it such that it samples values of
 144 \mathbf{x} that are close to the ones implic-
 145 itly maximizing $S(\mathbf{o}', \mathbf{o})$. At each it-
 146 eration t we sample a batch $\{\mathbf{x}_i \sim$
 147 $p(\mathbf{x}; \boldsymbol{\theta}_{t-1})\}_{i=1}^N$, obtain observations using the simulator $\{\mathbf{o}_i \sim \hat{p}(\mathbf{o}|\mathbf{x}_i)\}_{i=1}^N$ and calculate their re-
 148 spective scores $\{S(\mathbf{o}_i, \mathbf{o})\}_{i=1}^N$. The top qN samples with the best scores are selected, where q is a
 149 pre-selected quantile hyperparameter. We define $\delta = S(\mathbf{o}_{[qN]}, \mathbf{o})$, the score function value for the
 150 $[qN]$ -th sample. The complete CEM-inspired MACE objective is given by:

$$\arg \max_{\boldsymbol{\theta}} \sum_{i=1}^N \mathbf{1}_{S(\mathbf{o}_i, \mathbf{o}) \geq \delta} \log p(\mathbf{x}_i; \boldsymbol{\theta}). \quad (3)$$

151 Botev et al. [24] solve a stochastic program for the each iteration of $\boldsymbol{\theta}$. Instead, we optimize this
 152 objective via stochastic gradient descent (SGD) as described above. The full MACE algorithm is
 153 shown in Alg. 1.

154 3.2.1 Implementation

155 Although MACE is suitable in principle for any generative model, some considerations must be
 156 made for specific types of models.

157 **Autoregressive models** provide an explicit likelihood value for a sample \mathbf{x} using the chain rule:
 158 $p(\mathbf{x}) = \prod_i p(x_i | x_{i-1}, \dots, x_0)$. Therefore, they are straightforward to use with MACE by denoting
 159 the model weights as $\boldsymbol{\theta}$ and directly optimizing the objective in Eq. 3.

160 **VAEs** do not provide an explicit likelihood value $p(\mathbf{x})$. However, they are trained
 161 with a lower bound estimate of $p(\mathbf{x})$, namely the Evidence Lower Bound (ELBO)
 162 $\log p(\mathbf{x}|\mathbf{z}; \boldsymbol{\psi}) - D_{\text{KL}}(q(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi}) \| p(\mathbf{z}))$, where $p(\mathbf{x}|\mathbf{z}; \boldsymbol{\psi})$ is the VAE decoder parameter-
 163 ized by $\boldsymbol{\psi}$, $q(\mathbf{z}|\mathbf{x}; \boldsymbol{\phi})$ is the encoder parameterized by $\boldsymbol{\phi}$ and $p(\mathbf{z})$ is the prior distri-
 164 bution of the latent space parameterized by $\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z$. We can use the ELBO in Eq. 3
 165 as a lower bound of the log-likelihood term $\log p(\mathbf{x}; \boldsymbol{\theta})$, optimizing the VAE parameters:
 166 $\arg \max_{\boldsymbol{\psi}, \boldsymbol{\phi}} \sum_{i=1}^N \mathbf{1}_{S(\mathbf{o}_i, \mathbf{o}) \geq \delta} [\log p(\mathbf{x}_i | \mathbf{z}_i; \boldsymbol{\psi}) - D_{\text{KL}}(q(\mathbf{z}_i | \mathbf{x}_i; \boldsymbol{\phi}) \| p(\mathbf{z}))]$. We found the opti-
 167 mization of the entire set of VAE weights $\boldsymbol{\psi}, \boldsymbol{\phi}$ to be difficult. Instead, we use $\boldsymbol{\theta} = \{\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z\}$ as our
 168 tuned parameters, and keep the encoder and decoder weights frozen. This choice of $\boldsymbol{\theta}$ only affects the

²We compare MACE to this objective as a baseline; see experiment results in Sec. 4.1 and Sec. 4.2

169 second term of the objective, and specifically $p(\mathbf{z})$. The objective of this version of MACE, which
 170 we dub MACE-VAE, then becomes: $\arg \max_{\theta} \sum_{i=1}^N \mathbf{1}_{S(\mathbf{o}_i, \mathbf{o}) \geq \delta} [-D_{\text{KL}}(q(\mathbf{z}_i | \mathbf{x}_i) || p(\mathbf{z}; \theta))]$.

171 4 Experiments

172 MACE is designed to allow deep generative models to adapt quickly to new evidence. To demon-
 173 strate the potential of MACE in robotic applications, we consider several domains in which these
 174 models are used to capture complex data distributions and show that MACE leads to practical solu-
 175 tions, improving on the alternatives. In the subsections below, we describe two of these environments
 176 and how they fit our setting, followed by a summary of experiment results. A third environment,
 177 in which we use MACE to recover object point clouds from partial point cloud observations, is
 178 described in Appendix C, due to lack of space.

179 4.1 Inferring Object Shapes by Grasping

180 Inferring object shapes from tactile measurements [10] is vital when visual sensors are unavailable
 181 or limited. We investigate whether details about an object can be inferred using only the contact
 182 points between it and the robot gripper fingertips. Differing from the approach of Pastor et al. [10],
 183 which aims to classify objects into 36 classes, we consider an expressive prior distribution over
 184 possible shapes represented by a deep generative model.

185 **Dataset.** We use the “airplane” class from the
 186 ShapeNet [25] dataset as a representative collection
 187 of objects with complex shapes. This class contains
 188 4045 objects, represented as 2048-point PCs.

189 **Model.** The prior generative model $p(\mathbf{x}; \theta_0)$ is a VAE
 190 trained on full PCs of objects. We use MACE-VAE
 191 as described in Sec. 3.2.1. Our VAE architecture uses
 192 PointNet [26] in the encoder and a fully connected
 193 decoder, and is based on the implementation used in
 194 Daniel and Tamar [27]. Additional training details
 195 and hyperparameters can be found in Appendix A.

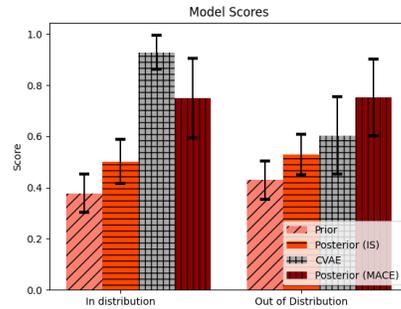
196 **Simulator.** For ease of implementation, we use a simple
 197 geometric simulator to calculate contact points.
 198 Details can be found in Appendix A.

199 **Score Function.** The score function for k -fingered
 200 grasps, aggregating distances between contact points
 201 and clipped to the range of $[0, 1]$, is defined as
 202
$$S(\mathbf{o}', \mathbf{o}) = \max \left(1 - \frac{1}{k} \sum_{j=1}^k \|p_j^{(\mathbf{o}')} - p_j^{(\mathbf{o})}\|, 0 \right),$$

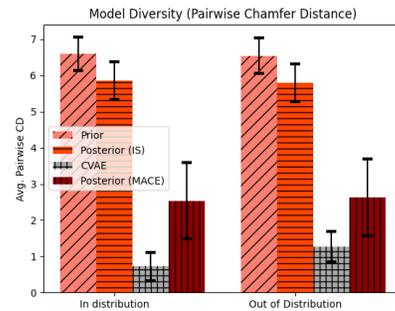
 203 where $p_j^{(\mathbf{o})}$ is the j -th contact point of observation \mathbf{o} .

204 Inferring Object Shapes by Grasping: Results

205 We adapt the distribution of airplane models by ob-
 206 taining a single observation \mathbf{o} representing contact
 207 points of $k = 5$ robot fingers with an unknown ob-
 208 ject. To evaluate the tuning process, we sample 49
 209 objects \mathbf{x}_i from the prior and another 49 from the posterior, and calculate the average score for the
 210 matching observations \mathbf{o}_i obtained from the simulator using $S(\mathbf{o}_i, \mathbf{o})$. In addition, we compute the
 211 pairwise Chamfer distances between every two objects in each sampled set and take their mean as a
 212 measure of sample diversity. Scores and sample diversity are presented in Fig. 1.



(a) Average scores



(b) Sample diversity

Figure 1: Quantitative results for the multi-fingered grasping experiments.

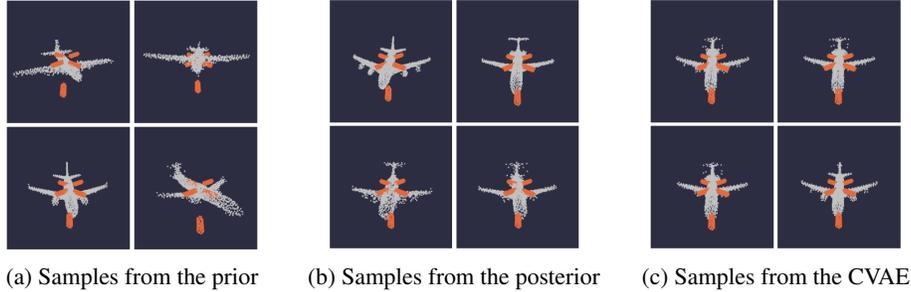


Figure 2: Tuning for the multi-finger grasping domain. Model samples are shown in white and finger positions and contact points of the given observation \mathbf{o} are represented by orange cylinders.

213 As a baseline, we conduct the same experiment using the importance sampling loss described in
 214 Eq. 2. We replace the log-likelihood term with the ELBO as in MACE-VAE (see Sec. 3.2.1), with
 215 the rest of the algorithm components as in Alg. 1. We find that this objective performs poorly
 216 compared to MACE (see Fig. 1). Moreover, optimizing it is an order of magnitude (up to $20\times$)
 217 slower than using MACE.

218 As an additional baseline, we train a CVAE conditioned on contact points calculated by grasping
 219 each training-set object in simulation. Albeit its advantage of fast amortized inference, the CVAE
 220 is limited to observations seen in its training set. We demonstrate this disadvantage in the out-of-
 221 distribution experiment below. We tune our model and compare results to the CVAE baseline using
 222 100 objects from the held-out test set of the ShapeNet “airplane” class, all with the same observation
 223 \mathbf{o} , in which the robot finger directions are the four diagonal corners of the xy plane, and a fifth along
 224 the x axis. Quantitative results can be seen in Fig. 1. The CVAE baseline outperforms the tuned
 225 posterior in scores, but produces a distribution with lower diversity.

226 In addition to the quantitative results, we present samples from the prior, posterior and CVAE models
 227 in Fig. 2. These showcase the diversity of the model tuned by MACE compared to the CVAE
 228 distribution. Additional samples and tuning hyperparameters can be found in Appendix A.

229 **Out-of-distribution experiment.** We run the entire set of experiments a second time using a differ-
 230 ent observation, with the fifth finger pointing along the opposite direction of the x axis. This is out
 231 of the joint distribution $p(\mathbf{x}, \mathbf{o})$ which the CVAE baseline was trained on. Consequently, its results
 232 greatly deteriorate. Conversely, the model tuned by MACE outperforms it both in diversity and in
 233 scores. Results can be seen in Fig. 1. Visuals of samples from the CVAE and the MACE-tuned
 234 posterior model with the new OOD observation can be viewed in Fig. 4 in Appendix A.

235 4.2 Inverse Kinematics with Obstacles

236 Inverse kinematics (IK) is the calculation of the configuration of robot joints given a desired pose in
 237 Cartesian space. IK calculation is an especially challenging optimization problem when obstacles
 238 are involved and has no closed-form solution in the general case. While previous work has attempted
 239 to learn IK using generative models [28, 29, 30], we focus on tuning a pre-trained IK model to con-
 240 sider novel obstacles. We view the obstacle-constrained IK problem as an inference problem, where
 241 the prior $p(\mathbf{x}; \theta_0)$ is a generative model trained to represent a distribution of joint configurations
 242 conditioned on the end-effector position³ when no obstacles are present. Note that this is a complex
 243 and multi-modal distribution which accounts both for self-collisions and for the conditioning on the
 244 desired pose. The observation is an obstacle configuration, and the posterior captures a distribution
 245 over non-colliding joint configurations.

246 **Dataset.** We collect 10M random valid configurations of a Franka Emika Panda 7-DoF robotic ma-
 247 nipulator using PyBullet physics simulation [31], and record their matching end-effector positions.

³The pose can also include the end-effector orientation; in this work we focus on position-only IK.

248 We collect these configurations in an environment with no obstacles present; therefore, valid con-
 249 figurations are ones which conform to the joint limits of the robot, and where the robot is not in
 250 collision with itself.

251 **Model.** We train an autoregressive model with joint positions generated sequentially, conditioned
 252 on the previous joints as well as the end-effector position: $p(q_1|\mathbf{p}_{ee}), p(q_2|q_1, \mathbf{p}_{ee})$ etc. Probability
 253 distributions for each joint are represented by Gaussian mixture models. Further architecture and
 254 training details can be found in Appendix B.

255 **Simulator.** We use open-source physical simulation environments, optionally with obstacles in the
 256 robot workspace. The simulated robot can be set to a specific joint configuration \mathbf{q} . The simulator
 257 returns whether the robot is in collision with itself or the obstacles, as well as the distance between
 258 the desired position and the actual end-effector position obtained by setting the robot to \mathbf{q} .

259 **Score Function.** We opt for a score function that penalizes collisions harshly, and therefore set
 260 $S(\mathbf{o}', \mathbf{o}) = 0$ if the robot is in collision in a given configuration. Otherwise, the score is proportional
 261 to the distance between the generated end-effector position and the desired position: $S(\mathbf{o}', \mathbf{o}) =$
 262 $\exp(-\|\mathbf{p}_{ee,desired} - \mathbf{p}_{ee,actual}\|)$.

263 Inverse Kinematics with Obstacles: Results

264 We run two experiments in this domain, with different types of obstacle in the workspace, both using
 265 the PyBullet simulation environment [31]. In the first experiment, the obstacle is a vertical window,
 266 with the desired end-effector positions located beyond it. A qualitative sample from the prior model
 267 $p(\mathbf{x}; \theta_0)$ (trained with no obstacles) can be viewed in Fig. 3a, where it is clear that many of the
 268 sampled configurations are in collision with the obstacle. Fig. 3b shows samples from the posterior
 269 model tuned with MACE, which almost never collide with the obstacle.

270 To show that the result does not depend
 271 on obstacle shape, we conduct a simi-
 272 lar experiment with a wall obstacle, with
 273 the target end-effector position behind
 274 it. Samples from the model tuned by
 275 MACE can be seen in Fig. 3c, again di-
 276 verse and non-colliding. We verify this
 277 result quantitatively by sampling 10 goal end-effector positions behind the wall, and tuning the
 278 model with the respective score functions. As a baseline, we also tune the model with the impor-
 279 tance sampling (IS) objective of Eq. 2. We report the mean scores over 1000 samples from the
 280 prior, the posterior tuned with MACE and the posterior tuned with the IS baseline in Table 1. We
 281 additionally report the success rate, calculated as the percentage of sampled configurations which
 282 are not in a collision state. The results clearly show improvement when tuning with MACE. Tuning
 283 hyperparameters and additional samples can be found in Appendix B.

Table 1: Inverse Kinematics Results

Model	Score	Success Rate
Prior	0.129 ± 0.015	0.132 ± 0.015
Posterior (MACE)	0.937 ± 0.079	0.941 ± 0.079
Posterior (IS)	0.317 ± 0.121	0.339 ± 0.122

284 Comparison to MoveIt Inverse Kinematics

285 The MoveIt [15] motion planning framework included with ROS has a standard IK service, used to
 286 infer goal positions for motion planning algorithms. While it is a powerful tool, we demonstrate that
 287 MACE can improve on its solutions where it struggles to find them quickly. We construct a scenario
 288 of a box in front of the robot, with the desired end-effector position inside it (see Fig. 3d). Using
 289 our prior model only (no tuning steps, for maximal speed), we sample 20 batches and test them
 290 for collisions in the IsaacGym [13] GPU-based simulator, using our score function. We take the
 291 configurations with the maximum score as our IK solution. In Table 2, we report calculation time
 292 as well as solution accuracy for our method compared to MoveIt. In addition, since MoveIt depends
 293 on the initial robot position for the IK calculation, we use the position sampled from our model
 294 as an initial position for MoveIt, thus reaping the benefits of both methods. Time and accuracy
 295 for this setting are reported in the third column of Table 2. Experiment details (including a MACE
 296 adaptation experiment for the box domain) and additional visual results are available in Appendix B.

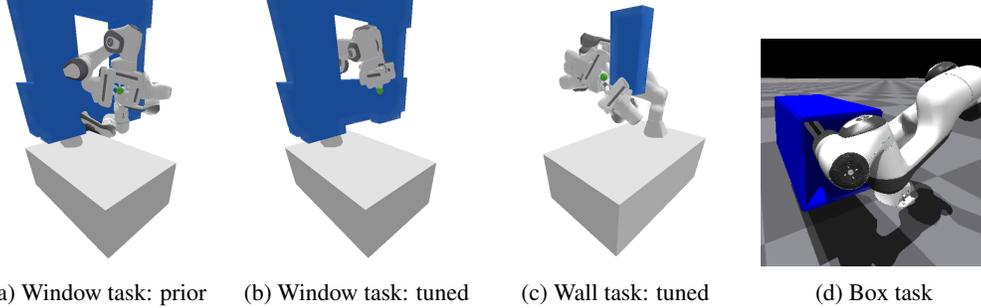


Figure 3: Tuning for the robot inverse kinematics domain. The obstacle is shown in blue, while the goal end-effector position is shown by a green ball. While Fig. 3a shows the prior distribution overlaid with the window obstacle, the same prior was used for all tasks (conditioned on the appropriate end-effector targets). Fig. 3b displays samples from the posterior distribution tuned with MACE for the window obstacle, while Fig. 3c shows the same for the wall obstacle. In both cases, the posterior rarely admits configurations colliding with the obstacles, while remaining diverse.

Table 2: Inverse Kinematics Comparison to MoveIt for the Box Task

MACE		MoveIt		MACE + MoveIt	
Time (s)	Acc. (cm)	Time (s)	Acc. (cm)	Time (s)	Acc. (cm)
0.106 ± 0.008	1.92 ± 3.01	1.553 ± 1.103	$< 10^{-5}$	0.641 ± 0.925	$< 10^{-5}$

297 5 Limitations

298 **Forward simulator.** A simulator that emits observations similar to the environment may not always
 299 be available, causing a sim-to-real gap which may deteriorate results. Approaches such as domain
 300 randomization [32, 18] may mitigate this problem.

301 **Inference speed.** In our experiments, MACE inference takes between 7 – 65 seconds (depending on
 302 the domain) which is still not fast enough for real-time inference applications. While the sequential
 303 nature of MACE optimization is an unavoidable computational limitation, code optimizations as
 304 well as faster hardware⁴ could dramatically speed up computation.

305 **Quality of the prior.** The quality of the tuned posterior depends greatly on the quality of the pre-
 306 trained deep generative model: if high-scoring samples have low probability under the prior, MACE
 307 may not find them. In our experiments, we found that deep generative models provide priors accurate
 308 enough for the domains we investigated.

309 6 Conclusion and Outlook

310 We presented MACE, a method for adapting deep generative models using the cross-entropy method,
 311 and demonstrated its usage for multiple robotic tasks. MACE allows the model to quickly adapt to
 312 previously unseen conditions while producing diverse posterior distributions. Our promising results
 313 for inverse kinematics show that deep generative models, when tuned appropriately using MACE,
 314 may help speed up robotic problems that are typically solved using non-learning based approaches.

315 In future work, we intend to explore ways to expedite the optimization process and improve the
 316 usability of MACE in robotic tasks. Additionally, in this work we only considered the inference
 317 problem. However, in a realistic scenario the agent may also have control over *which observations*
 318 *to acquire*. In this case, it would be interesting to extend MACE to an active sampling method. An-
 319 other related direction is to use MACE as an inference method for meta-RL, replacing the currently
 320 dominating CVAE-based approaches [12, 34].

⁴our experiments used unoptimized PyTorch [33] code and a single Nvidia GTX 1080 Ti GPU

References

- [1] P. N. Johnson-Laird. Mental models and human reasoning. *Proceedings of the National Academy of Sciences*, 107(43):18243–18250, 2010.
- [2] J. Redmon and A. Angelova. Real-time grasp detection using convolutional neural networks. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 1316–1322. IEEE, 2015.
- [3] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7559–7566. IEEE, 2018.
- [4] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.
- [5] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.
- [6] R. Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.
- [7] D. Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.
- [8] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International conference on LEarning representations*, 2013.
- [9] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.
- [10] F. Pastor, J. García-González, J. M. Gandarias, D. Medina, P. Closas, A. J. García-Cerezo, and J. M. Gómez-de Gabriel. Bayesian and neural inference on lstm-based object recognition from tactile and kinesthetic information. *IEEE Robotics and Automation Letters*, 6(1):231–238, 2020.
- [11] W. Yu, J. Tan, C. K. Liu, and G. Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.
- [12] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. *arXiv preprint arXiv:1910.08348*, 2019.
- [13] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.
- [14] R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*, volume 133. Springer, 2004.
- [15] D. Coleman, I. Sucas, S. Chitta, and N. Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.
- [16] J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate bayesian computational methods. *Statistics and computing*, 22(6):1167–1180, 2012.
- [17] M. Engel, O. Kanjilal, I. Papaioannou, and D. Straub. Bayesian updating and marginal likelihood estimation by cross entropy based importance sampling. *Journal of Computational Physics*, 473:111746, 2023.
- [18] F. Ramos, R. C. Possas, and D. Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019.

- 364 [19] N. Marlier, O. Brüls, and G. Louppe. Simulation-based bayesian inference for multi-fingered
365 robotic grasping. *arXiv preprint arXiv:2109.14275*, 2021.
- 366 [20] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–
367 1780, 1997.
- 368 [21] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional
369 generative models. *Advances in neural information processing systems*, 28, 2015.
- 370 [22] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep
371 networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.
- 372 [23] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL²: Fast rein-
373 forcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.
- 374 [24] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L’Ecuyer. The cross-entropy method for
375 optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier, 2013.
- 376 [25] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva,
377 S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint*
378 *arXiv:1512.03012*, 2015.
- 379 [26] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d
380 classification and segmentation. In *Proceedings of the IEEE conference on computer vision*
381 *and pattern recognition*, pages 652–660, 2017.
- 382 [27] T. Daniel and A. Tamar. Soft-introvae: Analyzing and improving the introspective variational
383 autoencoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern*
384 *Recognition*, pages 4391–4400, 2021.
- 385 [28] H. Ren and P. Ben-Tzvi. Learning inverse kinematics and dynamics of a robotic manipulator
386 using generative adversarial networks. *Robotics and Autonomous Systems*, 124:103386, 2020.
- 387 [29] C.-M. Hung, S. Zhong, W. Goodwin, O. P. Jones, M. Engelcke, I. Havoutis, and I. Posner.
388 Reaching through latent space: From joint statistics to path planning in manipulation. *IEEE*
389 *Robotics and Automation Letters*, 7(2):5334–5341, 2022.
- 390 [30] R. Bensadoun, S. Gur, N. Blau, T. Shenkar, and L. Wolf. Neural inverse kinematics, 2022.
- 391 [31] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics
392 and machine learning. 2016.
- 393 [32] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization
394 for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ*
395 *international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.
- 396 [33] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison,
397 L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.
- 398 [34] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. Efficient off-policy meta-
399 reinforcement learning via probabilistic context variables. In *International conference on ma-*
400 *chine learning*, pages 5331–5340. PMLR, 2019.
- 401 [35] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint*
402 *arXiv:1412.6980*, 2014.
- 403 [36] J. Lundell, F. Verdoja, and V. Kyrki. Robust grasp planning over uncertain shape completions.
404 In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages
405 1526–1532. IEEE, 2019.

- 406 [37] S. Chen, X. Ma, Y. Lu, and D. Hsu. Ab initio particle-based object manipulation. *arXiv*
407 *preprint arXiv:2107.08865*, 2021.
- 408 [38] Y. Sun, Y. Wang, Z. Liu, J. Siegel, and S. Sarma. Pointgrow: Autoregressively learned point
409 cloud generation with self-attention. In *Proceedings of the IEEE/CVF Winter Conference on*
410 *Applications of Computer Vision*, pages 61–70, 2020.
- 411 [39] L. Zhou, Y. Du, and J. Wu. 3d shape generation and completion through point-voxel diffusion.
412 In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5826–
413 5835, 2021.
- 414 [40] Y. Cai, K.-Y. Lin, C. Zhang, Q. Wang, X. Wang, and H. Li. Learning a structured latent space
415 for unsupervised point cloud completion. In *Proceedings of the IEEE/CVF Conference on*
416 *Computer Vision and Pattern Recognition*, pages 5543–5553, 2022.

417 Appendix

418 A Details for the Object Shape Inference Domain

419 Prior VAE Training Details

420 The prior used in this domain is a VAE trained on 2048-point PCs of ShapeNet “airplane” objects.
421 As it is based on the architecture used in Daniel and Tamar [27], we refer the reader to their work
422 for architectural details⁵. We train the VAE for 2000 iterations, augmenting the dataset with random
423 rotations around the vertical (z) axis in the range of $[-\frac{\pi}{4}, \frac{\pi}{4}]$. Both the encoder and decoder are
424 trained with the Adam optimizer [35], with a learning rate of 0.0005 and a batch size of 64. The
425 prior standard deviation is set to $\sigma_z = 0.2$, and the weighting parameters for the loss are set to
426 $\beta_{rec} = 50, \beta_{KL} = 1$. The latent space dimension is 128.

427 Grasping Simulator Implementation Details

428 To simplify implementation, we use a hand-crafted geometric simulator to calculate contact points
429 between a theoretical robot hand and object point clouds (PCs). We assume each finger is moved
430 along a vector pointing at the origin (which is located inside the object PC), and mark the point in
431 the PC furthest from the origin along this vector direction as the contact point. To simulate the width
432 of the finger, we consider points within a certain radius around the vector for contact calculation.
433 When grasping with k fingers, the observation $\mathbf{o} \in \mathbb{R}^{k \times 3}$ is the subset of contact points from the PC
434 \mathbf{x} .

435 Tuning Experiment Details

436 The prior $p(\mathbf{x}; \theta_0)$ is tuned for 2500 gradient steps, which takes approximately 25 seconds on a
437 single Nvidia GTX 1080 Ti GPU. We resample a new batch of $N = 256$ samples from the updated
438 model every $K = 32$ gradient steps, each taken on half of the batch due to memory constraints. We
439 use the Adam optimizer with learning rate 0.0002. We calculate the optimization objective with a
440 quantile value of $q = \frac{1}{16}$.

441 CVAE Baseline Hyperparameters

442 The CVAE baseline uses an architecture similar to the VAE prior model described above, with an
443 additional encoder to encode the condition contact points to a 128 dimensional latent $\mu_{prior}, \sigma_{prior}$.
444 In addition to its usage in the KL divergence loss, the prior mean μ_{prior} is injected into the decoder
445 in various layers. The CVAE baseline is mostly trained with the same hyperparameters as the VAE
446 described above, with two differences: $\beta_{KL} = 1$ and a learning rate of 0.0002.

447 Out-of-Distribution Experiment Visuals

448 Fig. 4 shows visual samples from the MACE-tuned prior and from the CVAE in the OOD experiment
449 (with the observation constituting a condition out of the distribution the CVAE was trained on).

450 Additional Model Samples

451 In this section we display additional samples for all of the distributions discussed in Sec. 4.1. All
452 visuals follow the same color scheme as in the main text, with samples shown in white and the grasp
453 positions represented by orange cylinders.

454 Fig. 5 shows samples from the pre-trained VAE prior. Figs. 6,7 display additional samples for the
455 first (in-distribution) experiment, for the posterior tuned by MACE and the CVAE baseline respec-
456 tively.

⁵See their public code at <https://github.com/taldatech/soft-intro-vae-pytorch>; we plan to release our code publicly at a later time.

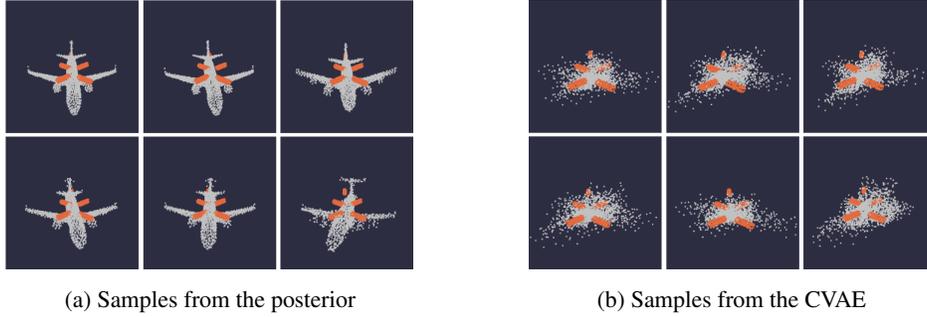


Figure 4: Samples from the posterior distribution tuned by MACE (left) and the CVAE baseline (right) when using an observation that is an OOD condition for the CVAE – note the gripper finger at the tail of the airplane. Results for MACE are similar to the in-distribution task, while the CVAE is unable to generate meaningful samples.

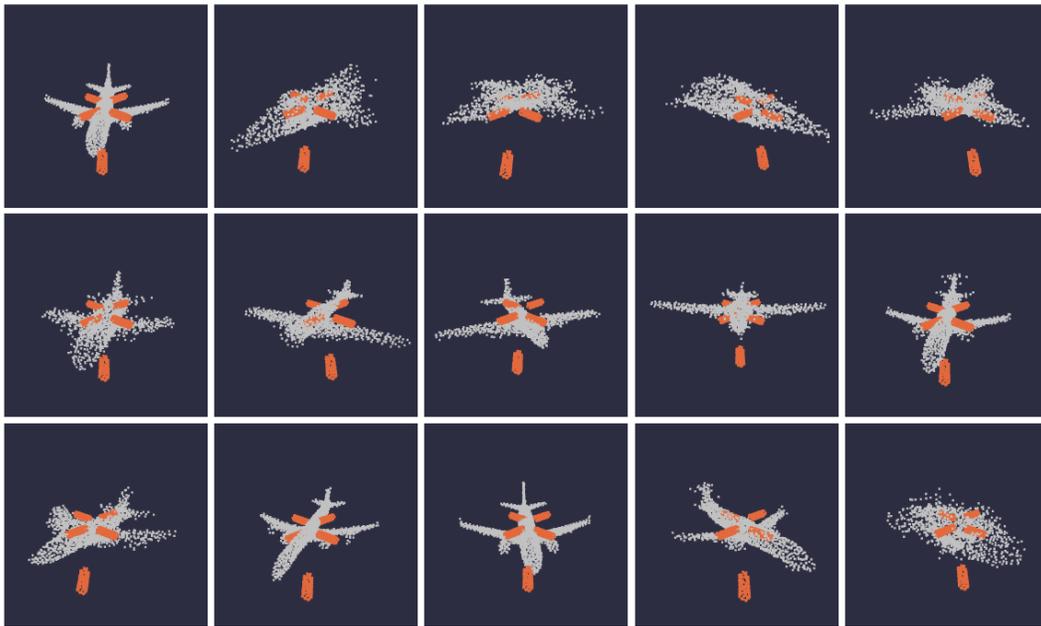


Figure 5: Samples from the prior

457 Figs. 8,9 display additional samples for the second (OOD) experiment, for the posterior tuned by
 458 MACE and the CVAE respectively.

459 B Details for the Inverse Kinematics Domain

460 Architecture of the Prior Model

461 As mentioned in Sec. 4.2, we train an autoregressive model to produce joint configurations condi-
 462 tioned on end-effector positions. We use 10M data points collected using the PyBullet simulator,
 463 and train the model end-to-end with the Adam optimizer in a supervised manner, using a maximum-
 464 likelihood objective over joint configurations.. Joint probabilities are represented by Gaussian mix-
 465 ture models with two components, each parameterized using a fully-connected NN with 5 layers of
 466 200 neurons, and Leaky ReLU activation functions.

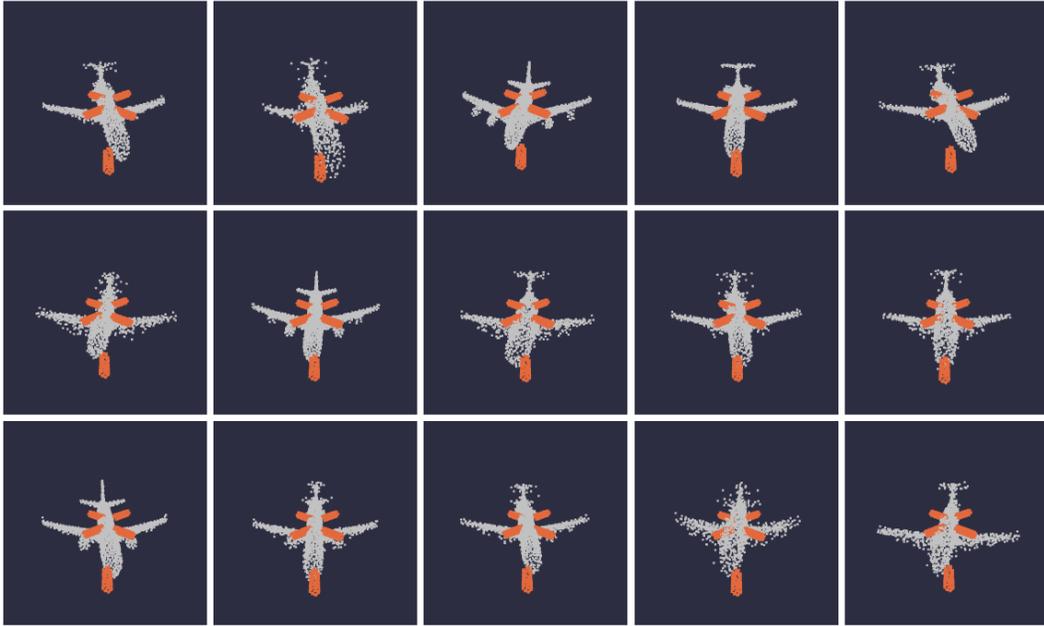


Figure 6: Samples from the posterior in the in-distribution experiment

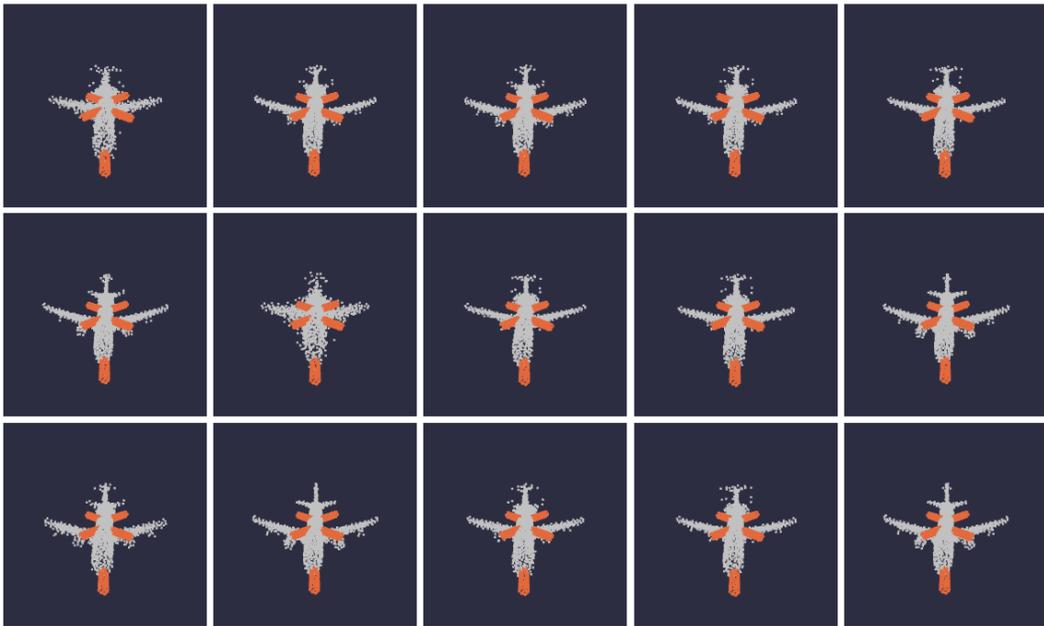


Figure 7: Samples from the CVAE in the in-distribution experiment

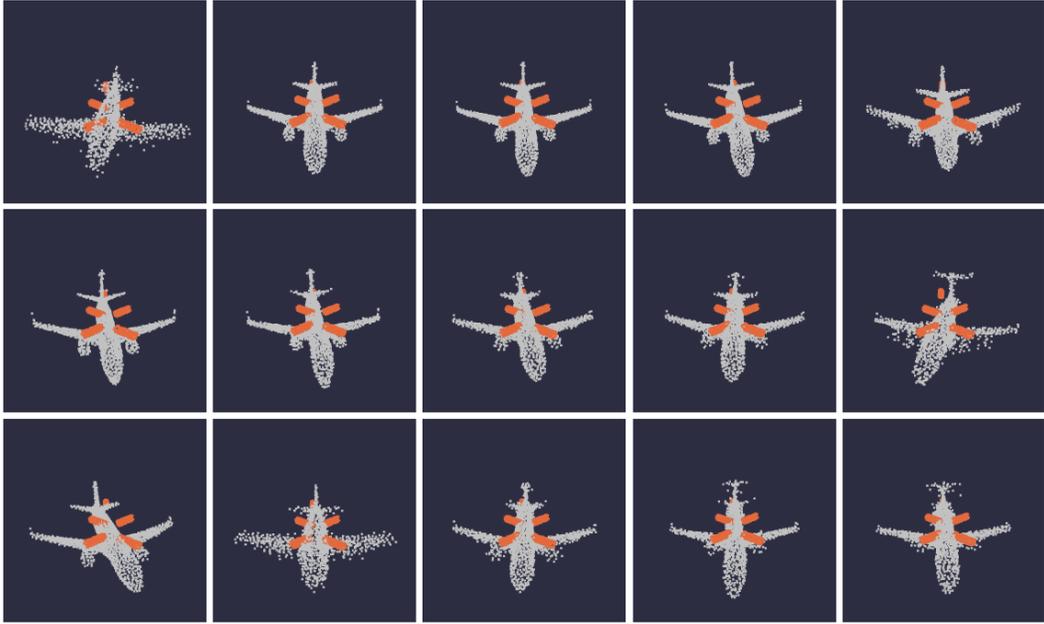


Figure 8: Samples from the posterior in the OOD experiment

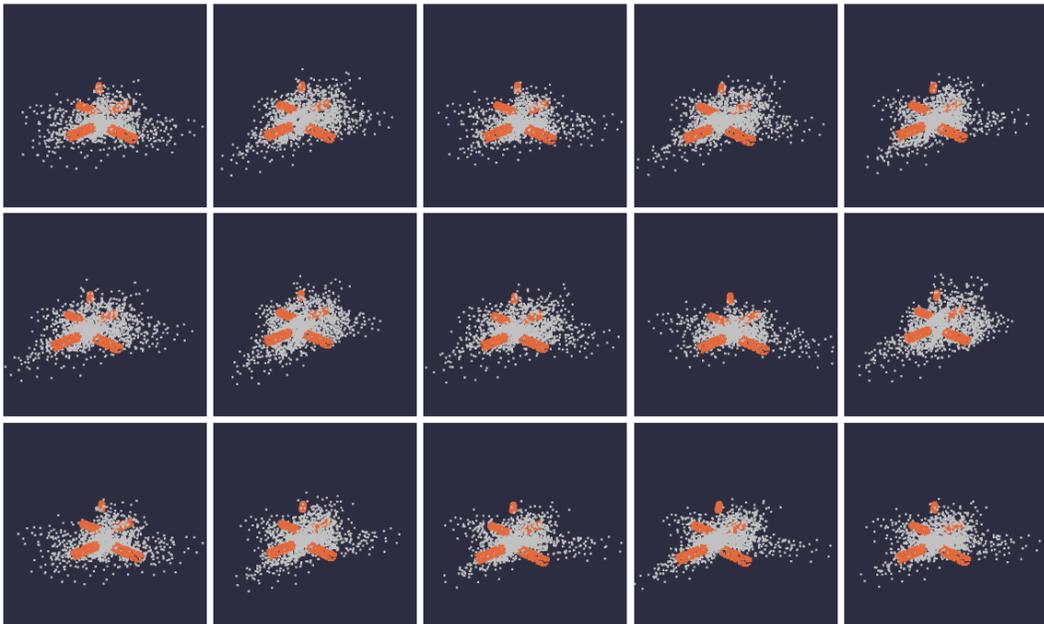


Figure 9: Samples from the CVAE in the OOD experiment

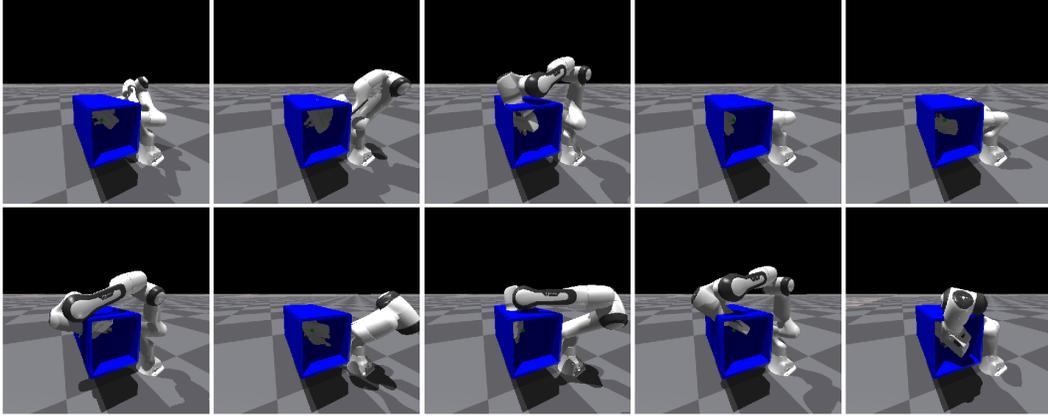


Figure 10: Samples from the prior overlaid with the box obstacle. Many of them collide with the walls of the box.

467 Experiment Details

468 **PyBullet Experiments.** The first two experiments described in Sec. 4.2 are conducted with the
 469 PyBullet physics simulation environment, with the wall and window obstacles. We tune the pre-
 470 trained prior for 1500 fine-tuning steps, which takes approximately 65 seconds on a single Nvidia
 471 GTX 1080 Ti GPU. We resample a new batch of $N = 64$ samples from the updated model every
 472 $K = 4$ gradient steps. We use the Adam optimizer with learning rate 0.00002. We calculate the
 473 optimization objective with a quantile value of $q = \frac{1}{16}$.

474 **MoveIt and IsaacGym Experiment.** For the box environment experiment and comparison to
 475 MoveIt, we use the same prior model, but instead use the GPU-based IsaacGym simulation en-
 476 vironment to expedite scoring the samples. To calculate the results described in Table 2 of Sec. 4.2,
 477 we sample 20 batches of 4096 configurations each, and test them for collisions in IsaacGym. Ob-
 478 taining the scores, we select the best configurations and report the mean and standard deviation of
 479 their distances from the goal in the accuracy column. The same configurations are used as initial
 480 positions for the “MACE + MoveIt” method in the third column, with the time constituting the total
 481 duration of sampling, testing for collisions with IsaacGym and finding solutions with MoveIt. The
 482 middle column reports times for MoveIt with a standard initial position. As MoveIt explicitly solves
 483 an optimization problem for the IK, its accuracy is very high; however, in some cases it takes much
 484 longer to find valid solutions.

485 **Tuning Experiment for the Box Domain.** In addition to the timing experiment, we conduct a
 486 tuning experiment with MACE on the box domain using IsaacGym. The experimental procedure is
 487 similar to the PyBullet experiments. We tune the model for 500 tuning steps, taking approximately
 488 10 seconds on a single Nvidia GTX 1080 Ti GPU with the faster IsaacGym simulator. We resample
 489 a batch of $N = 4096$ configurations every $K = 4$ gradient steps, and use a quantile of $q = \frac{1}{128}$.
 490 Fine-tuning is conducted using the Adam optimizer, with a learning rate of 0.0001. Samples from
 491 the prior can be found in Fig. 10, while samples from the tuned model can be seen in Fig. 11.

492 Additional Model Samples for the PyBullet Experiments

493 In this section, we provide additional samples for the distributions described in Sec. 4.2. Fig. 12 and
 494 Fig. 14 provide samples from the prior model, trained with no obstacles present in the workspace.
 495 This is the same distribution in both sets of samples, overlaid with different objects to show that
 496 many configurations collide with each of them.

497 Fig. 13 shows samples from the posterior tuned with MACE in the presence of the wall obstacle.
 498 Fig. 15 shows samples from the posterior tuned with MACE and the window obstacle.

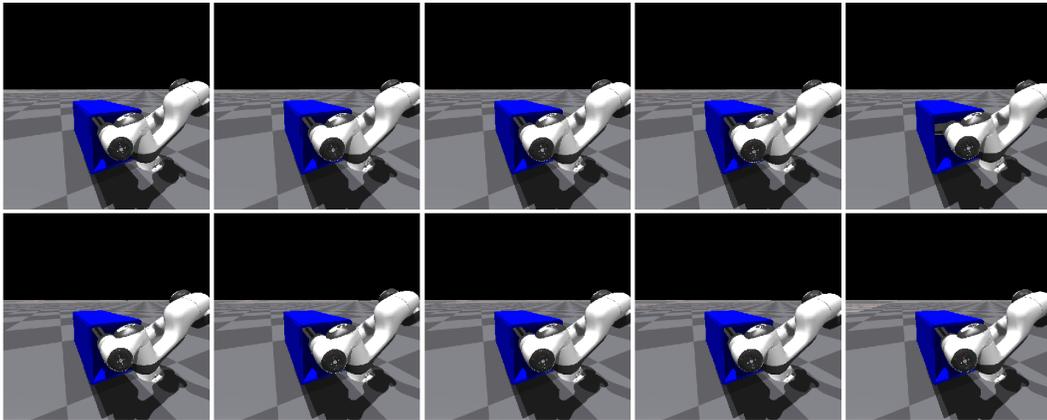


Figure 11: Samples from the posterior tuned with MACE to match the box obstacle.

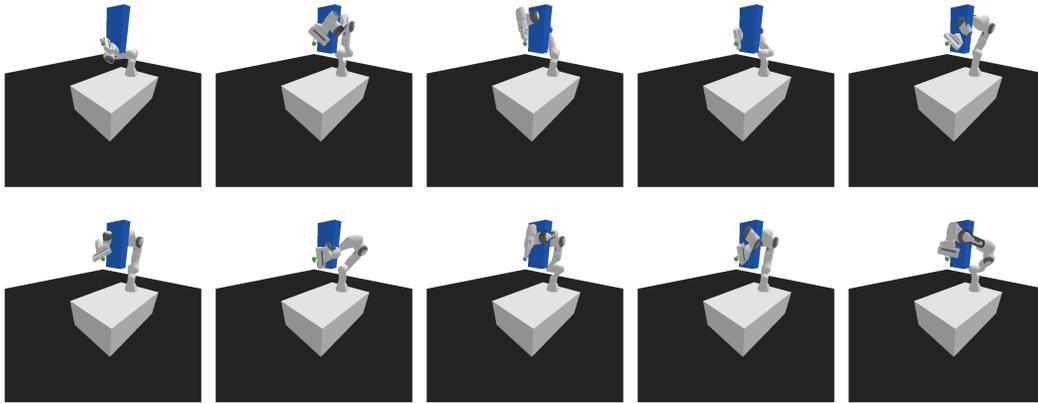


Figure 12: Samples from the prior overlaid with the wall obstacle.

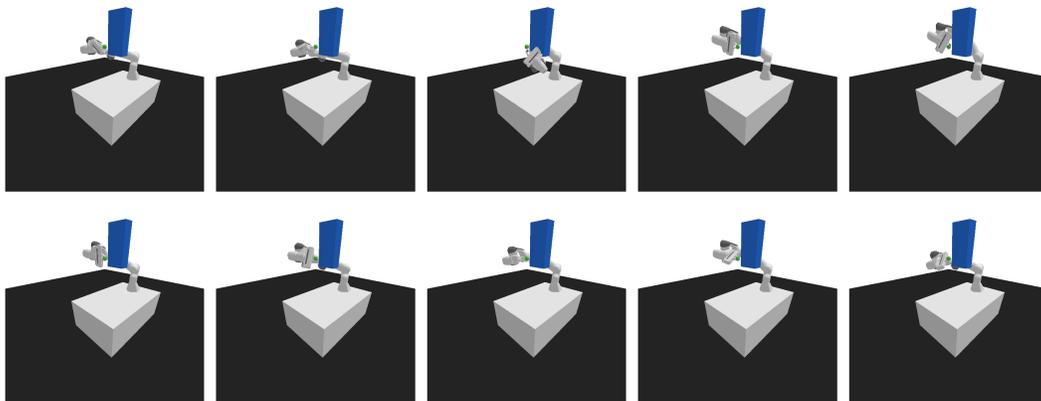


Figure 13: Samples from the posterior tuned to match observations of the wall obstacle.

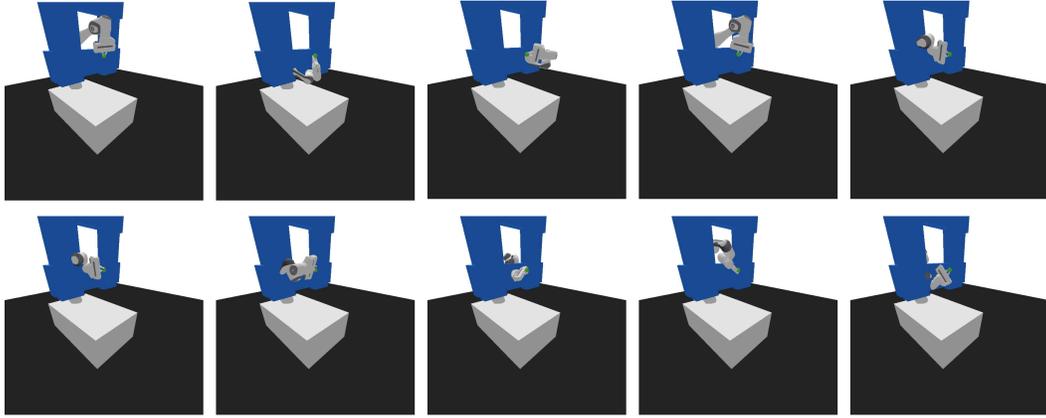


Figure 14: Samples from the prior overlaid with the window obstacle.

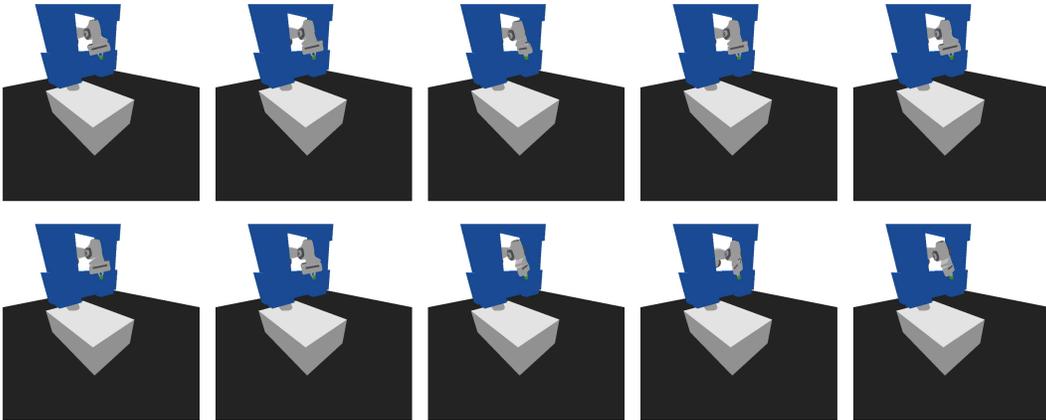


Figure 15: Samples from the posterior tuned to match observations of the window obstacle.

499 C The the Point Cloud Completion Domain

500 PC completion is an important component of manipulation pipelines, which allows robots to reason
 501 about their environment when partial information is available from sources such as depth sensors
 502 [36, 37]. Previous work typically focuses on scenarios in which a model can be faithfully recovered
 503 given the partial PC, i.e., when the dataset is small or the partial information is indicative of the
 504 object [38, 39, 40]. Instead, we consider a case where the posterior can be extremely multi-modal,
 505 and must therefore model a highly diverse distribution.

506 Given a partial PC as the observation \mathcal{o} , we infer a posterior distribution over possible full PCs \mathcal{x} .
 507 We include this domain as a proof-of-concept, and present qualitative results on a relatively simple
 508 dataset.

509 **Dataset.** We use a dataset of 10K symmetrical 3D boxes generated with random edge lengths,
 510 placed on the xy plane and centered around the z axis. Each PC consists of 2048 points, uniformly
 511 sampled on the box faces.

512 **Model.** We use a the same VAE architecture described in Sec. 4.1. The VAE is trained for 2000
 513 iterations with training samples augmented by random rotation around the vertical (z) axis in the
 514 range of $[-\pi, \pi]$. Both the encoder and decoder are trained with the Adam optimizer [35], with a
 515 learning rate of 0.0002 and a batch size of 64. The prior standard deviation is set to $\sigma_z = 1$, and
 516 the weighting parameters for the loss are set to $\beta_{rec} = 1, \beta_{KL} = 0.1$. The latent space dimension is
 517 128.

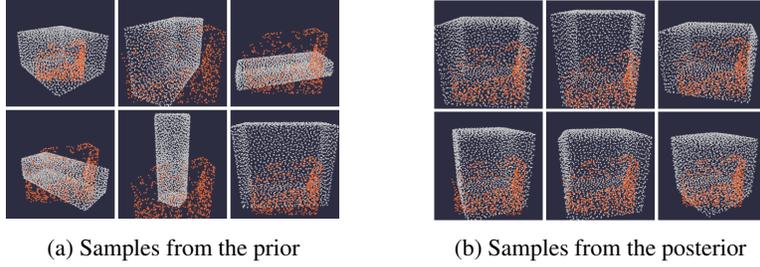


Figure 16: Tuning for the PC completion domain. Samples from the prior and posterior models are shown in white. Partial PC observation \mathbf{o} is overlaid over all samples in orange. While the prior model is extremely diverse and exhibits many different box sizes and rotations, the posterior tuned with MACE converges to samples which more closely match the evidence, while still producing a plausible distribution of objects.

518 **Simulator.** We require a simulator that can produce partial PCs given a full PC model. For this
 519 simple dataset, we obtain partial PCs by applying a random cut to each box, using a randomly
 520 sampled hyperplane. Note that this shape of the partial PC can fit a variety of different boxes,
 521 leading to a diverse posterior.

522 **Score function.** To measure similarity between PCs, $S(\mathbf{o}', \mathbf{o})$ is calculated using the Chamfer dis-
 523 tance between PCs \mathbf{o}' and \mathbf{o} . As suggested by Chen et al. [37], we find that calculating the distance
 524 to the top $k > 1$ nearest points produces better results than $k = 1$, and therefore use $k = 5$ when cal-
 525 culating the score function. Considering PCs \mathbf{x} and \mathbf{x}' with points labeled as $\{p_i\}_{i=1}^N$ and $\{p'_i\}_{i=1}^M$
 526 respectively, the original Chamfer distance is given by:

$$\text{CD} = \sum_{i=1}^N \min_{p'_i \in \mathbf{x}'} \|p'_i - p_i\|_2^2 + \sum_{i=1}^M \min_{p_i \in \mathbf{x}} \|p_i - p'_i\|_2^2.$$

527 The k -wise Chamfer distance replaces the min operation with a selection of the top- k nearest neigh-
 528 bors, denoted by the sets $\mathbf{x}^{(k)}$ and $\mathbf{x}'^{(k)}$:

$$\text{CD}_k = \frac{1}{k} \sum_{i=1}^N \sum_{p'_i \in \mathbf{x}'^{(k)}} \|p'_i - p_i\|_2^2 + \frac{1}{k} \sum_{i=1}^M \sum_{p_i \in \mathbf{x}^{(k)}} \|p_i - p'_i\|_2^2$$

529 To obtain scores in $[0, 1]$ with 1 being the maximum score, we set $S(\mathbf{o}', \mathbf{o}) = \exp(-\tau \text{CD}_k(\mathbf{o}', \mathbf{o}))$,
 530 where τ is a temperature parameter, set to $\tau = 0.1$ in our experiments.

531 Point Cloud Completion: Results

532 We use MACE-VAE (see Sec. 3.2.1) to tune the prior distribution parameters of the VAE latent space
 533 for 4000 fine-tuning steps, which take approximately 40 seconds on a single Nvidia GTX 1080 Ti
 534 GPU. We resample a new batch of $N = 256$ samples from the updated model every $K = 128$
 535 gradient steps, each taken on a batch of half of the samples. We use the Adam optimizer with
 536 learning rate 0.001. We calculate the optimization objective with a quantile value of $q = \frac{1}{32}$. Fig. 16a
 537 shows samples from the prior distribution $p(\mathbf{x}; \theta_0)$ overlaid with the partial PC observation \mathbf{o} ,
 538 while fig. 16b shows samples from the posterior model $P(\mathbf{x}; \theta_T)$ tuned with MACE. We observe
 539 that MACE can produce diverse completions of the partial PC. Additional samples can be found in
 540 Figures 17,18.

541 Additional Model Samples

542 Fig. 17 shows samples from the pre-trained VAE prior. Fig. 18 displays additional samples for the
 543 posterior tuned by MACE.

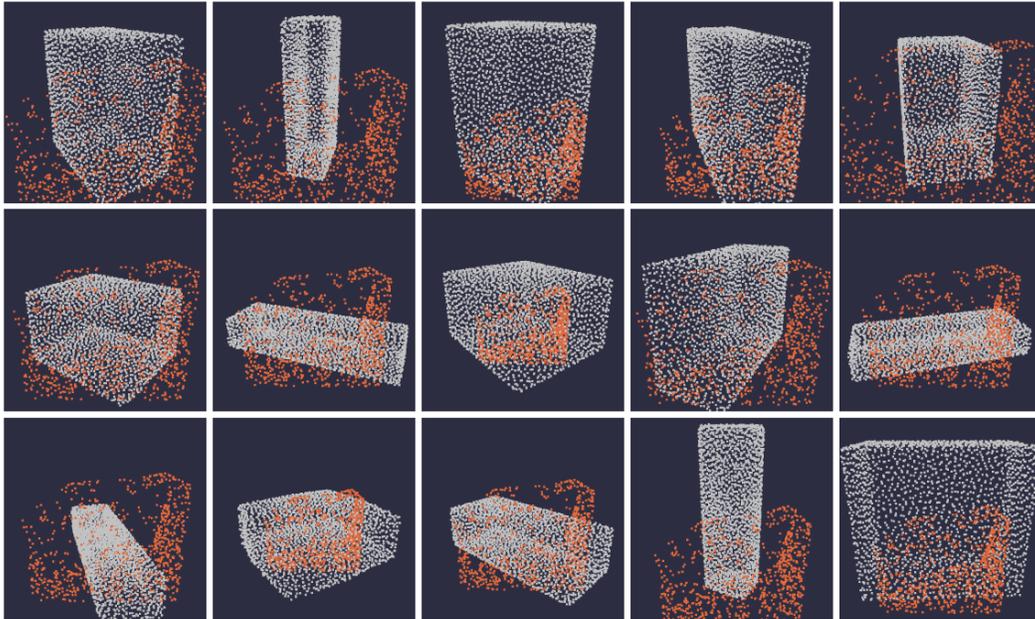


Figure 17: Samples from the prior

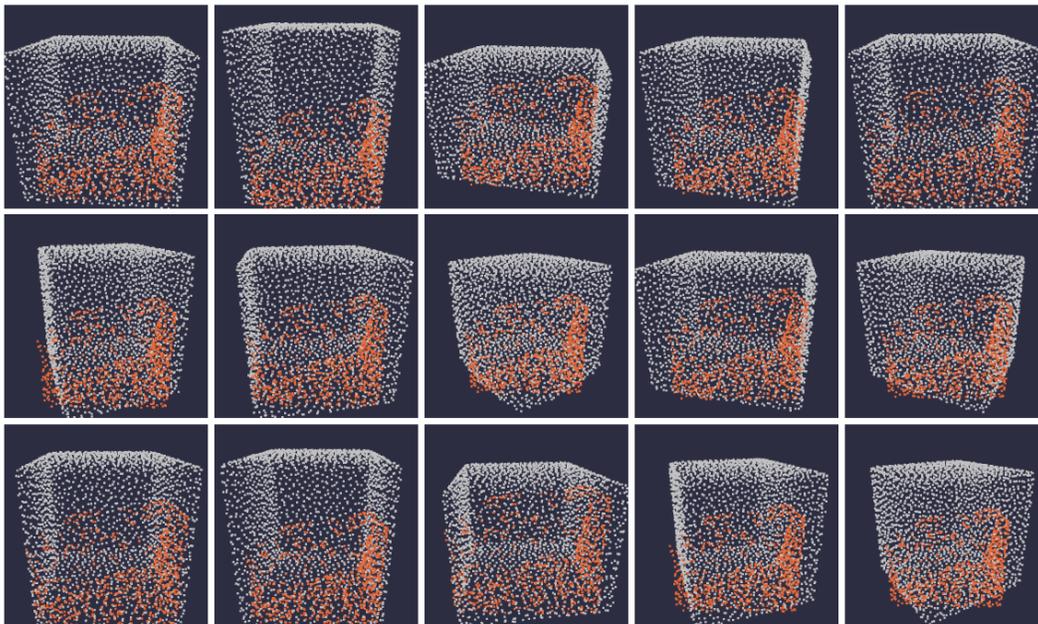


Figure 18: Samples from the posterior