A Dataset details

Real-world cubes Pfrommer et al. [2] provide a dataset of a real cube being tossed under frictional contact (setup shown in Figure 4a). The cube is 10cm across, made of acrylic, and tossed onto a wooden table. To produce more regular contacts, the cube's corners are coated with a gelatinous material. The cube's rotation R and position p are tracked at 148Hz using four AprilTags and TagSLAM [34]. Velocities are represented as the translational and body-frame angular velocity. While the original data contains 750 tosses, Pfrommer and Daniilidis [34] filtered them to 570 tosses whose initial toss directions were randomly rotated around the world z axis. Following [2], we split the dataset into 50% of the trajectories for training, 30% for validation, and 20% for testing.

Stiffness cubes Following Parmar et al. [10], we generated a set of three datasets in MuJoCo [19] for three cubes with different contact stiffnesses k using the parameters in Table 1. The contact stiffness k varies from 100 in the Soft condition, to 300 in the Medium condition, and 2500 in the Hard condition.

Table 1: Die Roll System Parameters (from [10])

Constant	Symbol	Value	Units
mass	m	0.37	kg
inertia	I	$6.167\mathrm{e}{-4}$	$\mathrm{kg\,m^2}$
side length	l	0.1	m
gravity	g	9.81	$\frac{\mathrm{m}}{\mathrm{s^2}}$
friction coefficient	μ	1	(none)
stiffness	k	(varies)	$\frac{\mathrm{N}}{\mathrm{kgm}}$
damping ratio	ζ	1.04	(none)
time-step	Δt	$6.74\mathrm{e}{-3}$	s

Initial states for the cube are sampled around a nominal state $x_{0,ref}$ as in `https://github.com/DAIRLab/ContactLearningBias` [10]. Using the generation code from `https://github.com/DAIRLab/ContactLearningBias`, we generate a set of 10,000 trajectories for each stiffness condition for training, with 1,000 trajectories for evaluation and testing.

Kubric objects We use Kubric [37] to generate a single-object variant of the Kubric MOVi-A dataset. The dataset contains 3 shapes (a bevelled cube, a cylinder and a sphere) with two sizes and two materials (rubber and metal, with corresponding friction and restitution coefficients). The objects have between 51 and 64 nodes. We generate a training set of 1000 trajectories with 96 steps each, as well as a test and validation set with 100 trajectories each.

B Model details

Model implementation and training The network architecture follows [3]. The networks ϵ, ρ, δ are 2-layer MLPs of width 128, with ReLu activations and residual connections. We apply LayerNorm to all MLP outputs, except for the decoder δ. All inputs and outputs of the GNN were normalized to zero-mean and unit variance based on dataset statistics.

The models were trained on 8 v100 GPU or TPUv4 devices for up to 1M steps, with the Adam optimizer, a learning rate of 10^{-4}, and a minibatch size of 64 per device. We apply Gaussian random-walk noise [16] with scale 3×10^{-4} to the node position inputs of the model.

Shape matching We implement the shape matching algorithm proposed by Müller et al. [36] (Section 3.2, Equations 7-8). The key idea is to first match initial object mesh M^U to the predicted nodes in M^t via a rigid transformation T. The predicted mesh M^t is then re-projected onto $T(M^U)$, thus preserving the initial rigid shape (Figure A.1).

Metrics We compute metrics for all experiments as in Pfrommer et al. [2]. That is, for each timestep t in each trajectory m of the test set, we compute

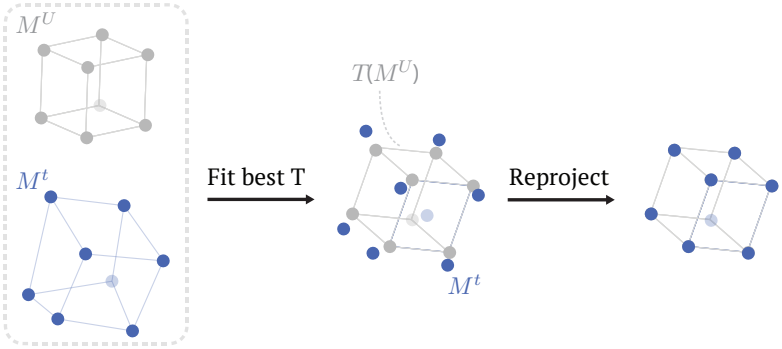

Figure A.1: Illustration of the shape-matching functionality: the initial object shape M^U is transformed (via T) to best fit the predicted nodes in M^t which are then re-projected back onto $T(M^U)$. This enforces the node predictions to preserve the initial shape [36].

1. the relative error norm $\frac{1}{w}||\hat{\mathbf{c}}_{t,m} - \mathbf{c}_{t,m}||_2$ of the predicted object center position $\hat{\mathbf{c}}$, where w is the object width,
2. the the absolute angle difference $|\text{angle}(\hat{\mathbf{R}}_{t,m}, \mathbf{R}_{t,m})|$ between the predicted object orientation $\hat{\mathbf{R}}$ and the ground truth $\mathbf{R}$, and
3. the floor penetration $\frac{1}{w}\max_i(\max(0, -z_{i,t,m}))$, where z_i is the z-coordinate of each corner node i.

All metrics are rollout metrics; that is, we compare model rollout from initial state to the ground-truth trajectory, and expect higher errors for later timesteps due to error accumulation. All metrics are averaged over both timesteps t and trajectories m. Since GNS outputs node positions, we obtain the object center and rotation using shape matching as described above. For all the plots we show the 95% confidence interval across 10 different seeds.

C Further analysis

C.1 Ablations on simulated cubes with different contact stiffness

We perform the same set of ablations from Figure 5 on the simulated cubes with different stiffness levels Figure A.2. The highest effect on performance is caused by the type of encoding: predicting abolute positions on node positions causes unstable predictions which greatly affects the positional and rotational errors. Using a GNS with a single message passing step achieves similar performance than the gated recurrent neural network (GRU) Parmar et al. [10]. For all cases, GNS with different architectural choices are not particularly sensitive to the contact stiffness with which contacts are modelled.

C.2 Additional ablations on the real cube

We perform an additional ablation on the real cube tossing data to study the effect of not encoding the initial object shape (Figure A.3). There is no significant difference between the best GNS* model and the GNS that excludes M^U and $\mathbf{d}_{ij}^U$ where shape matching is enabled, and only a minor effect when shape matching is disabled, indicating the explicit shape information is not needed and can be directly learned from data.

A.3(d) shows the ablations over four activation functions, commonly used for neural network training. We replace the activation functions in all MLPs used in the model, including encoder, decoder and networks for node and edge updates. ReLu, Softplus, Elu activation functions showed rollout positional error within standard deviations of each other, with lowest error for Softplus. While one could suppose that ReLu activation might be helpful for modelling sharp discontinuities, our results demonstrate that ReLu does not give any advantage over other 'smooth' activations, like Softplus or Elu.

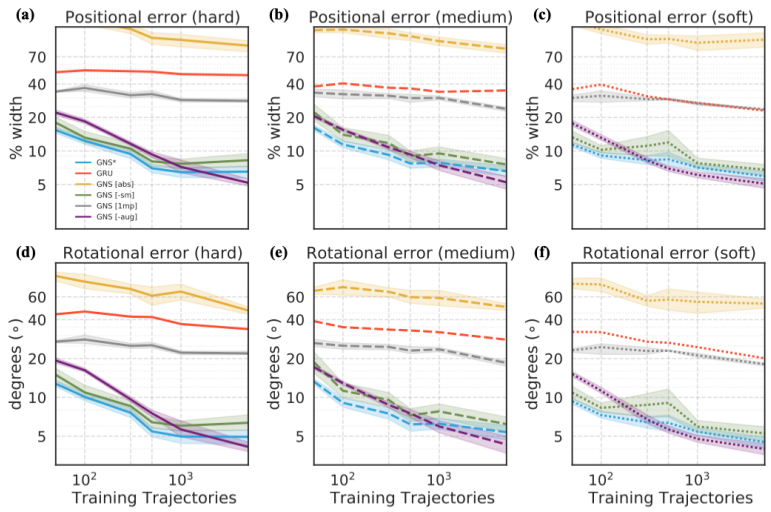

Figure A.2: Impact of different architectural choices for GNS on the simulated cubes with different stiffness for the Positional (a-c) and Rotational (d-f) errors. Lack of relative encoding (yellow) also causes unstable predictions (similar to the abltaion on the Real Cube from (Figure 5). Using a GNS with a single message passing step achieves similar performance than the gated recurrent neural network (GRU) Parmar et al. [10], however, still with less sensitivity to the stiffness type. The method is relatively robust to other architectural choices, with ablations decreasing performance but still allowing for better predictions than GRU.

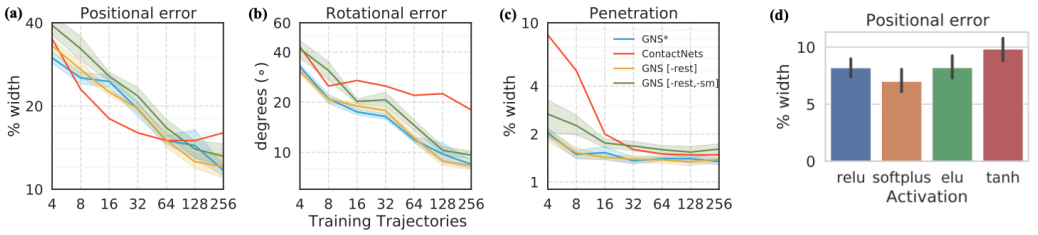

Figure A.3: (a-c) Additional real cube tossing ablations without encoding the rest mesh positions M^U in the graph, with and without shape matching. (d) Ablation over the activation functions: softplus activation results in the lowest positional error.

Data augmentation for ContactNets We also trained a ContactNets [2] model from `https://github.com/DAIRLab/contact-nets.git` with and without rotational data augmentation. In contrast to GNS, we do not observe a significant change in prediction accuracy: with data augmentation, the positional error is on average 8% lower, rotational error 7% higher, and penalty 12% lower, all within the 95% confidence window. We suspect the reason that ContactNets does not benefit from rotational data augmentation is that the inputs of the learned component are already aligned in the direction of the normal/tangential collision frame.

C.3 Additional qualitative analyses of learned models

Additional examples of discontinuities In Figure 1, we showed that a Graph Network Simulator, trained with only 256 real trajectories, could learn to capture hard dynamics discontinuities. This is not restricted to a single example – in almost all cases, the Graph Network Simulator correctly picks up on dynamics discontinuities in trajectories. We show 3 additional examples in Figure A.4.

Additional examples of sensitivity to initial angles We observed that trained Graph Network Simulators could learn to be acutely sensitive to initial conditions, such as falling left or right cor-

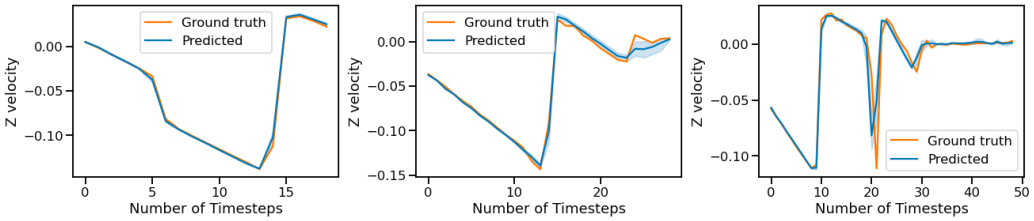

Figure A.4: Three further examples of discontinuities appearing in both the Graph Network Simulator and Ground truth rollouts across different numbers of timesteps. In all cases, the Graph Network Simulator picks up on hard discontinuities present in the ground truth rollout.

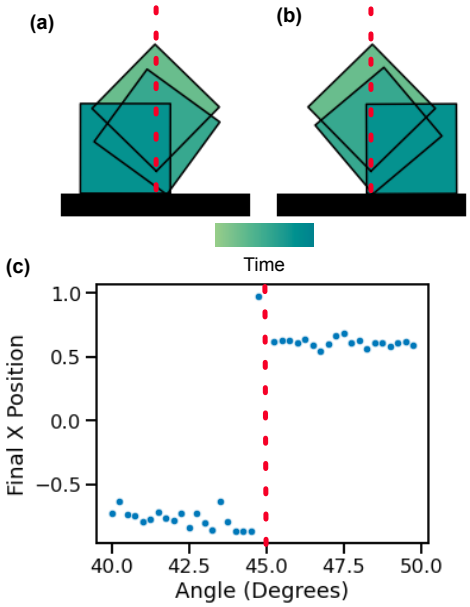

Figure A.5: Even if we train a Graph Network Simulator on just 256 real world trajectories, we still see a sharp contact discontinuity around $\theta = 45°$. For $\theta < 45°$, the cube falls left, while for $\theta > 45°$, the cube falls to the right.

rectly for fractions of a degree around $\theta = 45°$ (Figure 3). These models were trained on 8196 trajectories from the MuJoCo "stiffness cube (hard)" dataset. Here we perform the same experiment with a model trained on only 256 trajectories from the "real cubes" dataset (Figure A.5).

Despite being more noisy due to using less, real-world data, the Graph Network Simulator again captures the input discontinuity very well – falling left when $\theta < 44.75°$, and falling right when $\theta > 45°$. We notice that the prediction for $44.75°$ is incorrect here, likely due to dataset imbalance.

C.4 Comparing velocity vs. acceleration vs. position prediction with relative edge features

We ran two additional ablation experiments to demonstrate the advantages of predicting acceleration specifically vs. position or velocity. In Figure A.6 and Figure A.7, we compare models using relative position encoding but predicting velocity vs. position vs. acceleration. For both the Kubric dataset and the real cube tosses, predicting acceleration is statistically better than predicting velocity, although both are much better than predicting positions directly.

C.5 Ablating the mesh representation

To underscore the value of using mesh based representations of objects for modeling rigid body dynamics, we ran an additional ablation experiment where we eliminate the mesh representation,

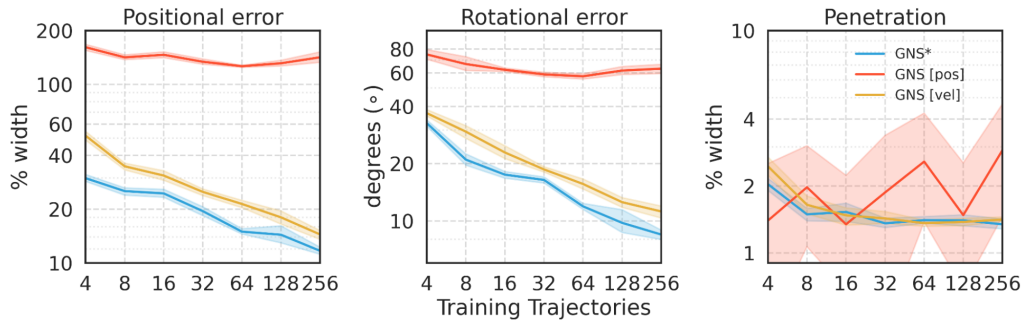

Figure A.6: Ablations on the Real Cube dataset for whether to predict acceleration (GNS*, blue), position (GNS [pos], red) or velocity (GNS [vel], yellow) while using relative edge features, all of which will then be integrated to a position-based loss. Acceleration performs best, although with less improvement over velocity prediction than position prediction.

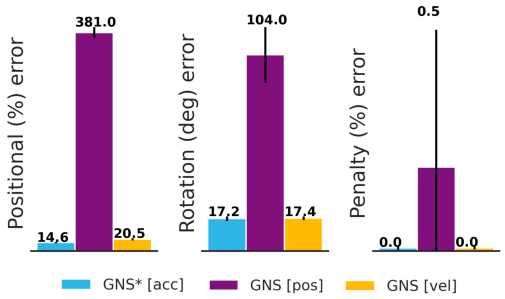

Figure A.7: Ablations on the Kubric MOVI-A dataset for whether to predict acceleration (GNS*, blue), position (GNS [pos], purple) or velocity (GNS [vel], yellow) while using relative edge features, all of which will then be integrated to a position-based loss. Acceleration performs best, although with less improvement over velocity prediction than position prediction.

but keep all other aspects of training intact (predicting acceleration, using the same network model, using the same history length, data augmentation, etc.) – fig:nograph. In this ablation, we predict the cube's translational and angular acceleration at the next time-point given a history of translation and rotation velocities, which is then converted to a position / rotation in the same manner as the main method. This ablation performs exceptionally poorly; displaying very high errors in the positions and rotations of the cube, but also very bad penetration scores with the floor. This confirms that GNNs excel at modeling these discontinuities at least in part because of the mesh representation of the object, rather than other architectural choices.

C.6 Sensitivity to history length

We tested the sensitivity of the model to different history lengths (1 and 2) on the Kubric and Real Cube datasets in Figure A.9. Using a history length of two minorly but consistently outperforms a history length of 1.

C.7 Sensitivity to hyperparameter K

We tested the model's sensitivity to the hyperparameter K by increasing the number of message-passing steps L while keeping $K = 1$ for the Kubric dataset. Small values of K can be compensated for by larger values of L.

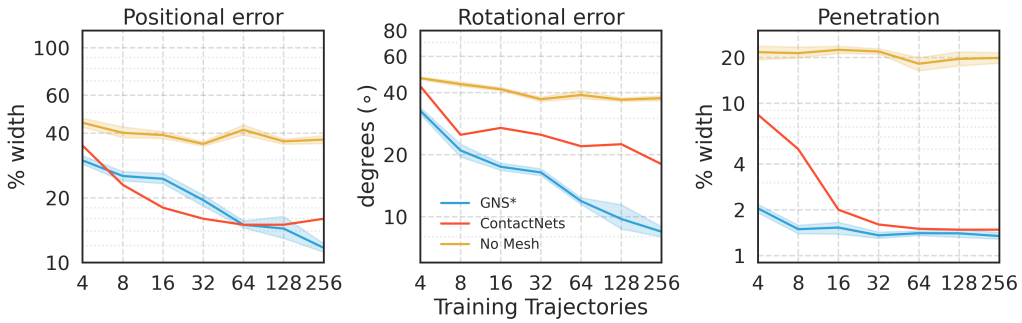

Figure A.8: Ablation experiment for the Real Cube dataset which removes the mesh representation of the object, and instead predicts the acceleration for the translation and rotation of the center of mass directly (No Mesh - yellow). This ablation performs poorly, and can no longer represent discontinuous dynamics effectively (see high Penetration errors).

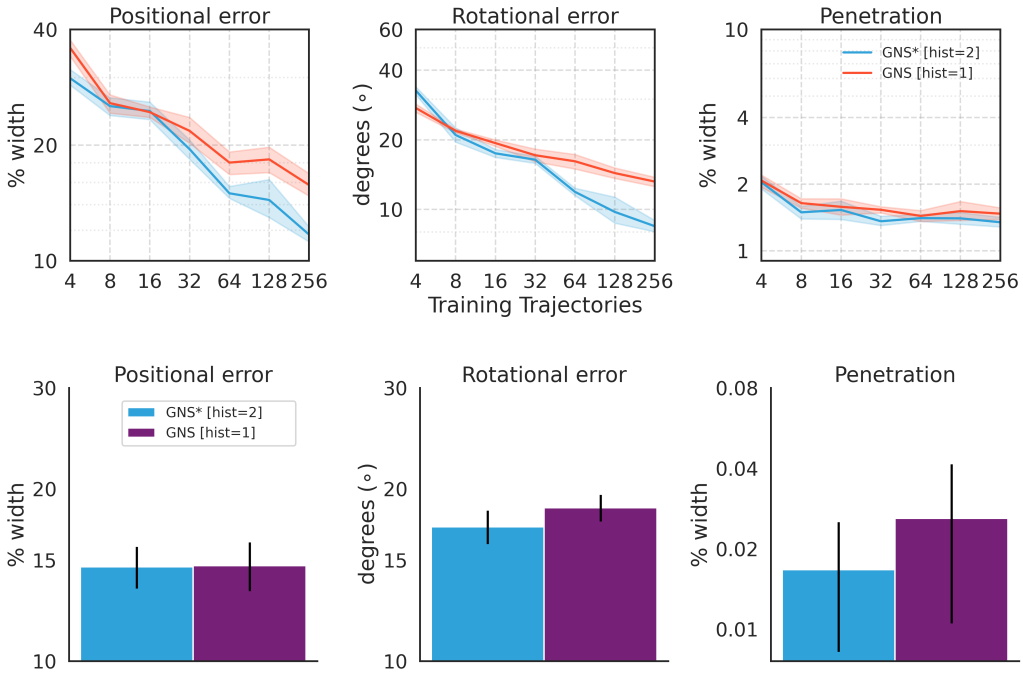

Figure A.9: Ablation testing the sensitivity to history length for the (top) Real Cube dataset and (bottom) Kubric dataset.

D Other metrics for Real Cube dataset

D.1 Training metrics

Additional training metrics for the Real Cube dataset can be visualized in Figure A.11.

D.2 Final frame comparisons

We compare the final frame errors to the ContactNets model for the Real Cube dataset. The analytical simulator results are taken from Acosta et al. [12] which does not provide code for replicating the training or evaluation, so it was not possible to recover final frame errors for Drake, MuJoCo and Bullet within the allotted rebuttal period. However, we note that the GNS* model still outperforms the *average* rotational error from the analytic simulators above 64 training trajectories. Since earlier

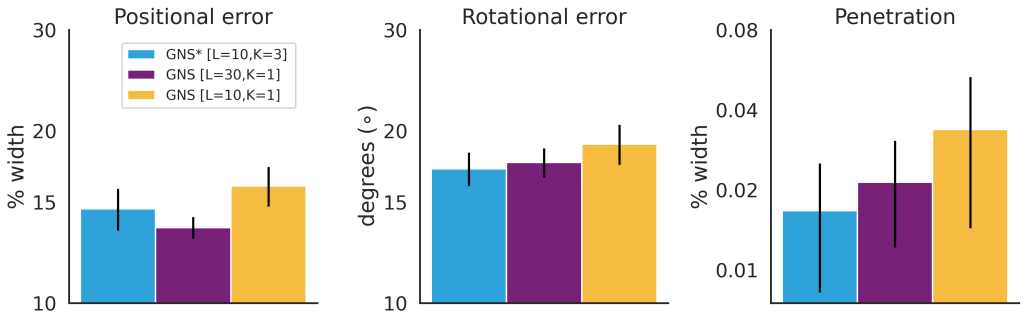

Figure A.10: Ablation testing the sensitivity to K for the Kubric dataset.

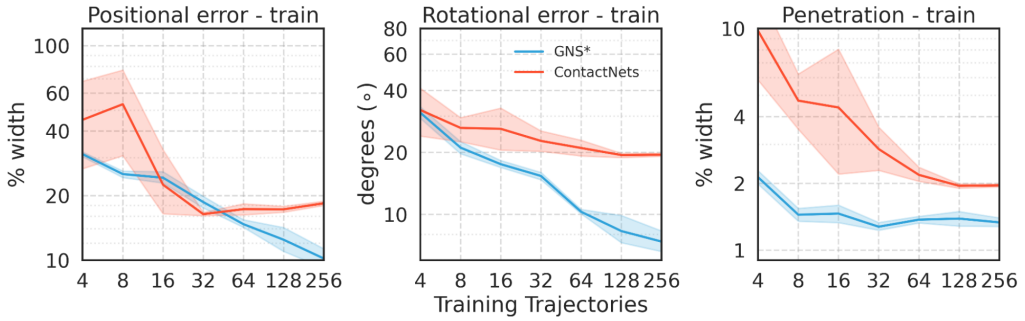

Figure A.11: Training errors for ContactNets and GNS* on the Real Cube dataset.

points along a trajectory will have lower errors than later points, this suggests that the GNS* model is still strongly outperforming the analytical models as well as the ContactNets model. For further evidence of this, we refer reviewers to https://sites.google.com/view/gnn-rigids/home which provides videos of the real cube being tossed. These can be compared to demonstrations from Acosta et al here: https://youtu.be/Ao6xJt4TpgU?t=16.

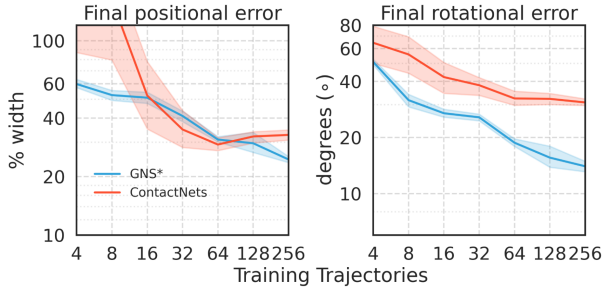

Figure A.12: Final frame errors for the Real Cube dataset for ContactNets and GNS*.

E Sensitivity to noise details

E.1 Shape / Mesh noise

We simulate "shape noise" by adding Gaussian noise with 0 mean and standard deviation σ individually to the nodes of the cube (of side length=2.0) as a way to model imperfect perception in Figure 7a. There is a graceful degradation of the positional and rotational error as the noise gets increased. However, even for an error of σ=0.2 on the perceived mesh, both rotational and position error are comparable to ContactNets when trained with 256 tosses.

E.2 Sensor noise

We simulate "sensor noise" by adding Gaussian noise with mean 0 and standard deviation σ independently to each transition in the dataset (x axis in Figure 7b). We scale the noise to the difference between the frames in order to simulate noise that depends on the speed of the moving cube.