A APPENDIX

B DATASET DETAILS

B.1 KUBRIC

Kubric dataset (Greff et al., 2022) consists of the rigid-body simulations of diverse 3D objects tossed simultaneously onto a large plane. The simulations are created using the Bullet simulator (Coumans, 2015). Kubric provides several datasets with increasing complexity of the object meshes: MoviA, MoviB, MoviC, etc. In this work, we train the models on MoviA and MoviB, as those already present a challenge to baseline models. We also provide the generalization experiments on MoviC.

Each trajectory in Kubric MoviA contains 3-10 objects of different sizes tossed toward the given position on the floor. In MoviA, only simple shapes are used: cubes (51 nodes), spheres (64 nodes), and cylinders (64 nodes). MoviB and MoviC contain simulations with 11 and 1030 objects respectively (the latter taken from the Google Scanned Objects Downs et al. (2022)), with between 51 and several thousand nodes.

The objects have two types of material properties: metal (friction 0.4, restitution 0.3, density 2.7) or rubber (friction 0.8, restitution 0.7, density 1.1) for MoviA or MoviB, For MoviC, the material parameters are the same for all objects (friction 0.5, restitution 0.5 and density 1.0). We append mass, friction and restitution to the features for each node of the object.

The Kubric dataset was originally created for perception tasks and includes images of the generated trajectories. In this work, we use only the information about the physical state, consisting of the rest position of the mesh, the object position and rotation quaterion. To obtain the state information, we re-generate the dataset using the github code `github.com/google-research/kubric`. For each of MoviA, MoviB and MoviC datasets, we use 1500 trajectories for training, 100 for validation and 100 for testing. Each trajectory consists of 96 time steps.

B.2 MIT PUSHING

Figure B.1: Objects used in the MIT pushing dataset. Figure credit to Yu et al. (2016).

The MIT Pushing Dataset introduced by Yu et al. (2016) consists of 6000 real-world pushes for 11 different objects (Figure B.1) along 4 different surfaces, resulting in a total of 264000 trajectories. The trajectories vary in that the pusher's velocity and acceleration can change, as well as the point of contact between the pusher and the object, and the angle between the pusher and the surface of the object where it makes contact. Object poses and pusher poses were recorded at a rate of 250 Hz, with each push unfolding over 5cm. We preprocess the data using the script provided by Kloss et al. (2022) at `github.com/mcubelab/pdproc`.

Trajectories from this dataset consist of the positions and rotations of the object and robotic pushing tip. To compare results to Kloss et al. (2022), we take the subset of trajectories which have a pusher acceleration of 0. Each trajectory is cut off after 0.5 seconds have passed, corresponding to 125 timesteps. From the full set of trajectories, we randomly choose 10000 trajectories across all objects from the "abs" surface material for training, and 1000 for testing. Of these 10000 trajectories, we sample different dataset sizes for training to examine FIGNet's sample efficiency.

For this dataset, we do not use static properties of objects in the node features, other than to indicate that nodes with mass 0 belong to the controlled pusher. However, we do subsample trajectories for training and rollouts to once every 3 timesteps, which stabilizes training. To compensate for errors in state estimation in the dataset, we found a larger node noise value was also critical (0.001). Because this dataset has a controlled pusher, we also do not predict the pusher's position. We predict only the

movement of the object mesh. Otherwise, all training details and architecture choices were identical to those used for the Kubric dataset.

C METHOD DETAILS

C.1 FULL MODEL DESCRIPTION

FIGNet uses an Encode-Process-Decode approach similar to Sanchez-Gonzalez et al. (2020); Pfaff et al. (2021). Crucially the encoder and processor are adapted to be able to support multiple node and edge types simultaneously. Also we introduce a new type of edge update function for message passing through face-face edges, which each have three sender and three receiver nodes, rather than just one. Note in our case face-face edges are always triangle-triangle edges, hence the three senders and receivers, but this approach would generalize to any other type of edges, such as triangle-point edges, tetrahedron-triangle edges, segment-triangle edges, etc. Also note that our approach is a strict generalization of message passing between pair of nodes, and when applied to point-point edges, it recovers all aspects of regular edge updates.

C.1.1 ENCODER

The encoder constructs the input graph $\mathcal{G} = (\mathcal{V}^{\text{M}}, \mathcal{V}^{\text{O}}, \mathcal{E}^{\text{M}}, \mathcal{E}^{\text{OM}}, \mathcal{E}^{\text{MO}}, \mathcal{Q}^{\text{F}})$ that represents the scene from the mesh M^t with 2 different node types, 3 different regular edge types, and 1 face-face edge type. Each node type has some features associated with it, which are encoded into fixed-size latent spaces.

Mesh nodes $\mathcal{V}^{\text{M}}$ represents the set containing each of the mesh nodes v_i^{M}, with input features $\mathbf{v}_i^{\text{M,features}} = [\mathbf{x}_i^t - \mathbf{x}_i^{t-1}, ..., \mathbf{x}_i^{t-h+1} - \mathbf{x}_i^{t-h}, \mathbf{p}_i, k_i, \mathbf{f}_i^t]$, where $\mathbf{x}_i^t$ is the position of the node at time t, $\mathbf{p}_i$ are static object properties, k_i is a binary "kinematic" feature that indicates whether the node is subject to dynamics (e.g. the moving objects), or its position is set externally (e.g. the floor), and $\mathbf{f}_i^t = k_i(\mathbf{x}_i^{t+1} - \mathbf{x}_i^t)$ is a feature that indicates how much kinematic nodes are going to move at the next time step. We use an MLP to encode all of these features into an initial latent representation for mesh nodes:

$$\mathbf{v}_i^{\text{M}} = \text{MLP}_{\mathcal{V}^{\text{M}}}^{\text{encoder}}(\mathbf{v}_i^{\text{M,features}})$$

Object nodes $\mathcal{V}^{\text{O}}$ represents the set containing each of the object nodes representing each rigid body v_i^{O}, with input features $\mathbf{v}_i^{\text{O,input}}$ analogous to those of the mesh nodes, using the center of mass of the object as the position for the object node. We use an MLP to encode all of these features into an initial latent representation for mesh nodes:

$$\mathbf{v}_i^{\text{O}} = \text{MLP}_{\mathcal{V}^{\text{O}}}^{\text{encoder}}(\mathbf{v}_i^{\text{O,input}})$$

Mesh edges $\mathcal{E}^{\text{M}}$ are bidirectional edges added between mesh nodes that are connected in the mesh. For each mesh edge $e_{v_{\text{s}}^{\text{M}} \to v_{\text{r}}^{\text{M}}}^{\text{M}}$ connecting a sender mesh node v_{s}^{M} to a receiver mesh node v_{r}^{M} we build edge features $\mathbf{e}_{v_{\text{s}}^{\text{M}} \to v_{\text{r}}^{\text{M}}}^{\text{M,features}} = [\mathbf{d}_{\text{rs}}, \mathbf{d}_{\text{rs}}^U]$, where $\mathbf{d}_{\text{rs}} = v_{\text{r}}^{\text{M}} - v_{\text{s}}^{\text{M}}$ is a vector of position differences between the nodes in the currently rotated mesh M^t (e.g. the positions of the nodes at the current of timestep, which are also affected by training noise); and $\mathbf{d}_{\text{rs}}^U$ is the position difference in the reference mesh M^U (which is static and independent of the dynamics). We use an MLP to encode all of these features into an initial latent representation for mesh nodes:

$$\mathbf{e}_{v_{\text{s}}^{\text{M}} \to v_{\text{r}}^{\text{M}}}^{\text{M}} = \text{MLP}_{\mathcal{E}^{\text{M}}}^{\text{encoder}}(\mathbf{e}_{v_{\text{s}}^{\text{M}} \to v_{\text{r}}^{\text{M}}}^{\text{M,features}})$$

Object-Mesh edges $\mathcal{E}^{\text{OM}}$ are unidirectional edges that go from each of the object nodes, to all of the mesh nodes that belong to that object. There is one object-mesh edge $e_{v_{\text{s}}^{\text{O}} \to v_{\text{r}}^{\text{M}}}^{\text{OM}}$ for each mesh node in $\mathcal{V}^{\text{M}}$. Input features $\mathbf{e}_{v_{\text{s}}^{\text{O}} \to v_{\text{r}}^{\text{M}}}^{\text{OM,features}}$ are analogous to the mesh edges,computed from the positions of the object and mesh nodes. We use an MLP to encode all of these features into an initial latent representation for mesh nodes:

$$\mathbf{e}_{v_{\text{s}}^{\text{O}} \to v_{\text{r}}^{\text{M}}}^{\text{OM}} = \text{MLP}_{\mathcal{E}^{\text{OM}}}^{\text{encoder}}(\mathbf{e}_{v_{\text{s}}^{\text{O}} \to v_{\text{r}}^{\text{M}}}^{\text{OM,features}})$$

Mesh-Object edges $\mathcal{E}^{\text{MO}}$ are unidirectional edges that go from each of the mesh nodes, to the object node representing the object it belongs to. There is also one mesh-object edge $e^{\text{MO}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}}$ for each mesh node in $\mathcal{V}^{\text{M}}$. Input features $\mathbf{e}^{\text{MO,features}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}}$ are analogous to the mesh edges, computed from the positions of the mesh and object nodes. We use an MLP to encode all of these features into an initial latent representation for mesh nodes:

$$\mathbf{e}^{\text{MO}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}} = \text{MLP}^{\text{encoder}}_{\mathcal{E}^{\text{MO}}}(\mathbf{e}^{\text{MO,features}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}})$$

Face-face edges $\mathcal{Q}^{\text{F}}$ are edges that connect two mesh faces belonging to different objects. A face-face edge $q^{\text{F}}_{\mathcal{F}^{\text{M}}_s \to \mathcal{F}^{\text{M}}_r}$ is added if any point of the sender face $\mathcal{F}^{\text{M}}_s$ is within the collision radius d_c from any point of the receiver face $\mathcal{F}^{\text{M}}_r$. This is a new type of edge, because from the point of view of the nodes of the graph, this is a "directed hyper edge", that connects the three mesh nodes of the sender mesh face $\mathcal{F}^{\text{M}}_s = (v^{\text{M}}_{\text{s1}}, v^{\text{M}}_{\text{s2}}, v^{\text{M}}_{\text{s3}})$, to the three mesh nodes of the receiver mesh face $\mathcal{F}^{\text{M}}_s = (v^{\text{M}}_{\text{r1}}, v^{\text{M}}_{\text{r2}}, v^{\text{M}}_{\text{r3}})$.

For each face-face edge, we build features, $\mathbf{q}^{\text{F,input}}_{\mathcal{F}^{\text{M}}_s \to \mathcal{F}^{\text{M}}_r} = [\mathbf{d}_{\text{rs}}, [\mathbf{d}_{s_j}]_{j=1,2,3}, [\mathbf{d}_{r_j}]_{j=1,2,3}, \mathbf{n}_{\text{r}}, \mathbf{n}_{\text{s}}]$. The features are defined with respect to the "closest collision points" between the two faces p_{s} (on the sender face $\mathcal{F}_s$), and p_{s} (on the receiver face$\mathcal{F}_r$). Geometrically, the closest point in the face might be either inside of the face, on one of the triangle edges or at one of the nodes. Then the features are defined as follows: (1) the relative vector between the closest points $\mathbf{d}_{\text{rs}} = \mathbf{p}_{\text{r}} - \mathbf{p}_{\text{s}}$ for the two faces, (2) the spanning vectors of three nodes of the sender face $\mathcal{F}_s$ relative to the closest point at that face $\mathbf{d}_{s_i} = \mathbf{x}_{s_i} - \mathbf{p}_s$, (3) the spanning vectors of three nodes of the receiver face $\mathcal{F}_r$ relative to the closest point at that face $\mathbf{d}_{r_i} = \mathbf{x}_{r_i} - \mathbf{p}_r$, and (4) the face normal unit-vector of the sender and receiver faces $\mathbf{n}_{\text{s}}$ and $\mathbf{n}_{\text{r}}$, pointing towards the outside of the object.

We use an MLP, followed by a reshape operation, to encode all of these features into an initial latent representation for each face-face edge as **three** vectors, one for each receiver node: $\mathbf{q}^{\text{features}}_{\mathcal{F}_s \to \mathcal{F}_r}$ into three face-face edge latent vectors, one for each receiver node $\mathcal{F}^{\text{M}}_s = (v^{\text{M}}_{\text{r1}}, v^{\text{M}}_{\text{r2}}, v^{\text{M}}_{\text{r3}})$ of each face-face interaction:

$$Q_{\mathcal{F}^{\text{M}}_s \to \mathcal{F}^{\text{M}}_r} = [\mathbf{q}_{j,\mathcal{F}_s \to \mathcal{F}_r}]_{j=1,2,3} = \text{reshape}(\text{MLP}^{\text{encoder}}_{\mathcal{Q}^{\text{F}}}(\mathbf{q}^{\text{features}}_{\mathcal{F}^{\text{M}}_s \to \mathcal{F}^{\text{M}}_r})))$$

Having one edge vector per edge receiver node will be a crucial aspect of the face-face edge update and mesh node update. Crucially, for each edge, and before computing features (2) and (3), we sort the nodes of the sender face $\mathcal{F}^{\text{M}}_s = (v^{\text{M}}_{\text{s1}}, v^{\text{M}}_{\text{s2}}, v^{\text{M}}_{\text{s3}})$ and receiver face $\mathcal{F}^{\text{M}}_s = (v^{\text{M}}_{\text{r1}}, v^{\text{M}}_{\text{r2}}, v^{\text{M}}_{\text{r3}})$ as function of the distance to the closest collision point in the corresponding face. This achieves permutation equivariance of the entire model, w.r.t. to the order in which the sender, and receiver nodes of each face are specified.

C.1.2 Message passing and iterative processor

The goal of one step of message passing in the processor is to update all latent representations in the encoded graph based on neighborhood information. This includes mesh edge latents $\mathbf{e}^{\text{M}}_{v^{\text{M}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}$, object-mesh edge latents $\mathbf{e}^{\text{OM}}_{v^{\text{O}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}$, mesh-object edge latents $\mathbf{e}^{\text{MO}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}}$, face-face edge latents $Q_{\mathcal{F}^{\text{M}}_s \to \mathcal{F}^{\text{M}}_r}$, object node latents $\mathbf{v}^{\text{O}}_i$, and mesh node latents $\mathbf{v}^{\text{M}}_i$, to produce updated (indicated as "prime") versions of those: $\mathbf{e}^{\text{M}}_{v^{\text{M}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}{}'$, $\mathbf{e}^{\text{OM}}_{v^{\text{O}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}{}'$, $\mathbf{e}^{\text{MO}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}}{}'$, $Q_{\mathcal{F}^{\text{M}}_s \to \mathcal{F}^{\text{M}}_r}$, $\mathbf{v}^{\text{O}\prime}_i$ and $\mathbf{v}^{\text{M}\prime}_i$. From a high level perspective the update occurs in two stages: (1) run an edge update (also referred to as "message function") on all of the different edge types (mesh edges, object-mesh edges, mesh-object edges and face-face edges), gathering information about the nodes adjacent to the edge, and (2) run a node update for each node type (mesh nodes and object nodes), aggregating information from edges of different types adjacent to the nodes.

The previous paragraph describes a single layer of message passing, but following a similar approach to Sanchez-Gonzalez et al. (2020); Pfaff et al. (2021) this layer can then be applied iteratively, as many times as necessary, with either shared or unshared neural network weights. Also similar to those we also add residual connections (sum the input to each layer to the output of the layer) at each layer, to improve gradient flow during training. Our processor consists of 10 steps of message passing, with unshared weights.

Regular edge updates are used to update the mesh edge latents $\mathbf{e}^{\text{M}}_{v^{\text{M}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}$, object-mesh edge latents $\mathbf{e}^{\text{OM}}_{v^{\text{O}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}$, mesh-object edge latents $\mathbf{e}^{\text{MO}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}}$. Each edge is updated following the edge update approach from Battaglia et al. (2018):

$$\mathbf{e}'_{v^X_{\text{s}} \to v^Y_{\text{s}}} = \text{MLP}^{\text{processor}}_{\mathcal{E}^{\text{XY}}}([\mathbf{e}_{v^X_{\text{s}} \to v^Y_{\text{s}}}, \mathbf{v}^X_s, \mathbf{v}^Y_r])$$

where X, Y may take the role of mesh (M) or object (O) nodes, depending on the type of edge.

Face-face edge updates are used to update the face-face edge latents $Q_{\mathcal{F}^{\text{M}}_s \to \mathcal{F}^{\text{M}}_r}$. The approach to update each face-face edge is similar to regular edges, except that now each edge has three latent vectors, three senders and three receivers, and we also need to produce three output vectors:

$$Q_{\mathcal{F}_s \to \mathcal{F}_r}{}' = [\mathbf{q}'_{j,\mathcal{F}_s \to \mathcal{F}_r}]_{j=1,2,3} = \text{reshape}(\text{MLP}^{\text{processor}}_{\mathcal{Q}^{\text{F}}}([[\mathbf{q}_{j,\mathcal{F}_s \to \mathcal{F}_r}, \mathbf{v}^{\text{M}}_{s_j}, \mathbf{v}^{\text{M}}_{r_j}]_{j=1,2,3}]))$$

noting that, this does not update the three latent vectors independently, but all nine input latent vectors $[[\mathbf{q}_{j,\mathcal{F}_s \to \mathcal{F}_r}, \mathbf{v}^{\text{M}}_{s_j}, \mathbf{v}^{\text{M}}_{r_j}]_{j=1,2,3}]$ contribute to all three updated face-face edge latent vectors $Q_{\mathcal{F}_s \to \mathcal{F}_r}{}'$.

Object node updates are used to update the object node latents $\mathbf{v}^{\text{O}}_i$. Each node is updated following the node update approach from Battaglia et al. (2018), aggregating all of the updated edge latents (or messages) for mesh-object edges that are adjacent to that object node:

$$\mathbf{v}^{\text{O}\prime}_i = \text{MLP}^{\text{processor}}_{\mathcal{V}^{\text{O}}}\big([\mathbf{v}^{\text{O}}_i, \sum_{\forall e^{\text{MO}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}} / v^{\text{O}}_{\text{r}} = v^{\text{O}}_i} \mathbf{e}^{\text{MO}}_{v^{\text{M}}_{\text{s}} \to v^{\text{O}}_{\text{r}}}{}'])$$

Mesh node updates are used to update the mesh node latents $\mathbf{v}^{\text{M}}_i$. Mesh nodes may receive information from object-mesh edges, mesh edges, and face-face edges, so to update each mesh node, information of adjacent edges to that node is aggregated for each of the edge types:

$$\mathbf{v}^{\text{M}\prime}_i = \text{MLP}^{\text{processor}}_{\mathcal{V}^{\text{M}}}\big([\mathbf{v}^{\text{M}}_i, \sum_{\forall e^{\text{OM}}_{v^{\text{O}}_{\text{s}} \to v^{\text{M}}_{\text{r}}} / v^{\text{M}}_{\text{r}} = v^{\text{M}}_i} \mathbf{e}^{\text{OM}}_{v^{\text{O}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}{}', \sum_{\forall e^{\text{M}}_{v^{\text{M}}_{\text{s}} \to v^{\text{M}}_{\text{r}}} / v^{\text{M}}_{\text{r}} = v^{\text{M}}_i} \mathbf{e}^{\text{M}}_{v^{\text{M}}_{\text{s}} \to v^{\text{M}}_{\text{r}}}{}', \sum_{\forall q^{\text{F}}_{\mathcal{F}_s \to \mathcal{F}_r} / \mathcal{F}^{\text{M}}_r[j] = v^{\text{M}}_i} \mathbf{q}'_{j,\mathcal{F}_s \to \mathcal{F}_r}]\big)$$

where the second and third terms correspond to regular edge aggregation (just like in the object node update), and the last term sums over the set of face-face edges for which v^{M}_i is one of the receivers and $\mathbf{q}'_{j,\mathcal{F}_s \to \mathcal{F}_r}$ selects the specific face-face edge vector corresponding to v^{M}_i as a receiver, i.e. from $Q'_{\mathcal{F}_s \to \mathcal{F}_r}$, selects the first, second, or third vector, depending on whether v^{M}_i, is the first, second or third vector in that receiver face (recall it is always sorted by distance to the closest point in the face, so the model remains permutation equivariant).

C.1.3 Decoder

Similar to Sanchez-Gonzalez et al. (2020); Pfaff et al. (2021), the goal of the decoder is to produce output features. In our model we only require output features for the mesh nodes v^{M}_i. We produce this by applying an MLP decoder from the latent space of the mesh nodes after the processor, which outputs a finite-difference approximation to the acceleration:

$$\mathbf{a}_i = \text{MLP}^{\text{decoder}}_{\mathcal{V}^{\text{M}}}(\mathbf{v}^{\text{M}}_i)$$

C.2 Other model details

Norm features For all relative spatial feature vectors $\mathbf{d}$, we also also concatenated their norm $|\mathbf{d}|$ as part of the inputs.

Training noise To make models stable for long rollouts while training on one-step data, we use the same strategy from Sanchez-Gonzalez et al. (2020); Pfaff et al. (2021)to train with random walk noise in the inputs, asking the model to correct for noise in the input velocities.

Predicted targets Similar to Sanchez-Gonzalez et al. (2020); Pfaff et al. (2021) our models predicts a finite-difference acceleration that is used to update the position $\mathbf{x}_i^{t+1} = \mathbf{a}_i + 2\mathbf{x}_i^t - \mathbf{x}_i^{t-1}$.

Normalization Similar to Sanchez-Gonzalez et al. (2020); Pfaff et al. (2021) we normalized all inputs and targets to zero-mean unit-variance. The loss is computed in the normalized space of the targets. This means our entire model and loss is agnostic to the scale of the data.

MLPs We use MLPs with 2 hidden layers, and 128 hidden and output sizes (except the decoder MLP, with an output size of 3). All MLPs, except for those in the decoder, are followed by a LayerNormBa et al. (2016) layer.

Optimization All models are trained to 1M steps with a batch size of 128 across 8 TPU devices. We use Adam optimizer, and an an exponential learning rate decay from 1e-3 to 1e-4.

D BASELINE DETAILS

D.1 MUJOCO AND BULLET SIMULATORS FOR REAL-WORLD MIT PUSHING DATASET

For the MIT Pushing Dataset we compare with the MuJoCo (Todorov et al., 2012) and Bullet (Coumans, 2015) simulators. While the objects have a provided mesh geometry, the physical properties of this real-world system, as well as the optimal internal simulator parameters, are unknown and thus must be estimated by system identification.

We construct this system identification problem as a black-box optimization procedure as follows. The objective we minimize is the sum of the relative translation error and relative rotation error of the object being pushed. To measure this objective we use a 50-trajectory subset of the full training dataset; making predictions for these 50 trajectories takes 5-10 minutes for each simulator. Using more trajectories for fitting did not yield improved results. As our optimizer, we choose Bayesian optimization with Gaussian processes as implemented by Vizier (Golovin et al., 2017). We perform 500 trials of optimization, then take the best-performing hyperparameters and evaluate them on a 100-trajectory validation set.

For MuJoCo we model the externally-controlled pusher as a non-physical motion capture body attached via an equality constraint to the physically-modeled pusher geometry. This allows the simulator to impute velocities for the geometry which are unavailable when using motion capture positions directly, making the physics stabler and more realistic. We find that modeling contacts with three degrees of freedom and neglecting the differential effects of rotational and rolling friction gives strictly superior results and so perform our final optimization using this setting. We use the expensive but more accurate RK4 integrator rather than the default Euler integrator for greater precision in modeling hard contact. We search over the space of scalar friction coefficients $k \in [0, 5]$ for all three objects, as well as over the number of physics substeps $\{1, 10, 100, 1000\}$. The best-performing solution has $k_{\text{object}} = 1.91$, $k_{\text{pusher}} = 0.41$, and $k_{\text{floor}} = 0.0$, and uses 1000 physics substeps per data timestep. For Bullet we create a fixed constraint for the tip and directly update the parent location of the fixed constraint with the positions of the tip given by the input trajectories. We search over the space of lateral friction coefficients $k \in [0, 5]$ for all three objects, as well as over the number of engine substeps $\{0, 1, 2, 3, 4, 5\}$. The best-performing solution has $k_{\text{object}} = 0.28$, $k_{\text{pusher}} = 0.0$, and $k_{\text{floor}} = 1.106$, with 0 substeps.

D.2 MESHGRAPHNETWORKS (MGN)

To make the comparison more informative and fair, we reimplement MeshNets to refer to a model with the same "bells and whisles" to ours, except for two joint ablations of important differences with (Pfaff et al., 2021).

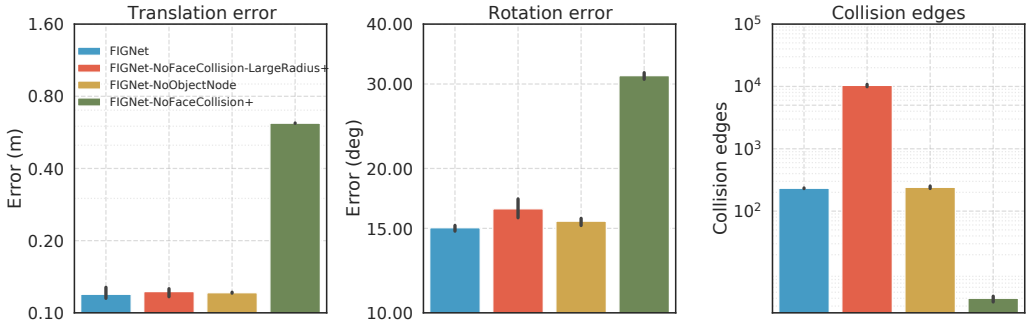


Figure E.1: Model ablations on Movi-A dataset. Replacing the face-face collisions with the node-node collisions makes the performance deteriorate the most. Models with * use an implicit floor representation, while models with $^+$ use a subdivided floor.

Remove object nodes (FIGNet-NoObjectNode). The ablation consists of removing the object nodes, as well as all of the object-mesh/mesh-object edges from the graph.

Node based collisions (FIGNet-NoFaceCollision). The ablation consists of computing collisions according to the distance between the mesh nodes, rather than the faces, and replacing the face-face edges, by regular mesh-mesh edges, referred to as "world edges" in (Pfaff et al., 2021). Unlike MeshGraphNets, this model has the per-object node, similarly to FIGNet. In order to give this ablation a chance, we (1) use a larger collision radius d_c (LargeRadius) and (2) subdivide the two gigantic triangles that make up the floor into smaller triangles of similar in size to the collision radius (1.5), or (3) when (2) becomes too we use an "implicit floor" by adding an extra feature to the nodes indicating if they are within the collision radius distance d_c from the floor and by how much. We also provide some results from applying these ablations independently.

D.3 DPI-Reimplemented

To make the comparison more informative and fair, we use DPI-Reimplemented to refer to a model with the same "bells and whisles" to ours, except for three joint ablations of important differences with (Li et al., 2019b).

Object-level predictions (FIGNet-DPILoss) The ablation consists of: (1) use a decoder for the object nodes, to produce a velocity/acceleration, for the position and rotation quaternion of the center of mass of the object, (2) used that prediction to update the mesh nodes positions, (3) compute the effective mesh node acceleration, and (4) use this to build the loss.

Particles (FIGNet-Particles) We replaced the mesh data by a dense particle representation of the object, and use those particles as nodes. As there aren't any faces we also use node-based collisions. Note in this case, we still keep the central object node, following (Li et al., 2019b) which would correspond to a $k = 1$ level hierarchy in their model.

We also provide some results from applying these ablations independently.

E Ablation results

To motivate the importance of the per-object nodes and face-face collisions, we provide the ablations of our model where we remove either of these properties from the model.

Figure E.1 and Figure E.2 demonstrate the ablation on Movi-A and Movi-B datasets respectively. Removing the face-face collisions, but keeping the same collision radius (NoFaceCollision model) makes the performance deteriorate on both Movi-A and Movi-B, presumably because the current collision radius does not allow to capture all the nodes that are necessary to resolve the collision.

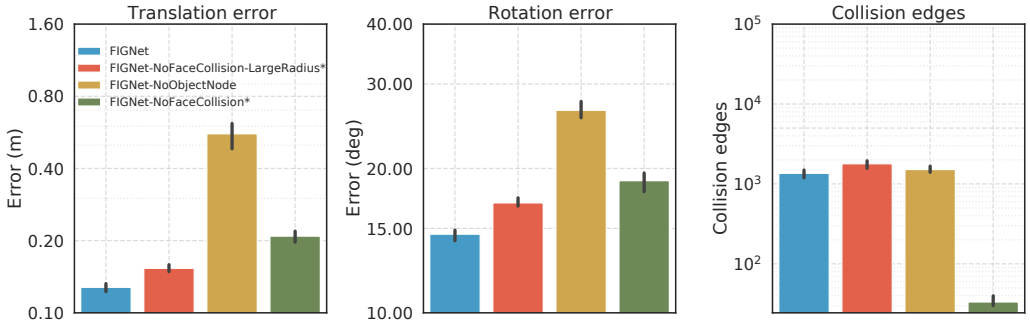


Figure E.2: Model ablations on Movi-B dataset. Removing the object node and replacing face-face collisions with node-node drastically affects the performance, making it worse. Models with * use an implicit floor representation, while models with + use a subdivided floor.

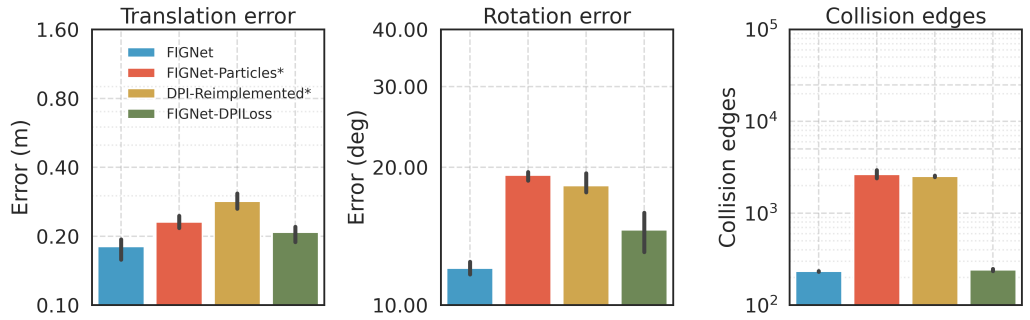


Figure E.3: Model ablations towards DPI (Li et al., 2019b) on Movi-A dataset. While the loss formulation has little effect, replacing the mesh with densely sampled particles creates huge issues with memory requirements, and makes overall translation and rotation error worse. Models with * use an implicit floor representation.

Increasing the collision radius (NoFaceCollision-LargeRadius) restores the performance on Movi-A, but it is not sufficient on Movi-B to match the performance of FIGNet.

Next, we consider the ablation of removing the per-object node from FIGNet. On Movi-A this modification does not affect the performance as much, presumably because the small number of nodes per object in Movi-A is relatively small and it does not require additional message-passing through the object node. However, on Movi-B we observe much higher translation and rotation errors compared to FIGNet and other baselines. This result highlights the importance of object-level node in the complex datasets like Movi-B, as provides a "shortcut" the message-passing between the nodes of the object.

Note that the FIGNet-NoFaceCollision and FIGNet-NoFaceCollision-LargeRadius baselines replace the face-face message-passing with node-node interactions. They suffer from the same issues related to floor parameterizations as MeshNets, as desribed in the main text. Therefore we use the parameterization with subdivided floor for Movi-A, and implicit floor for Movi-B for these models, similarly to MeshNets (see explanation for Figure 5a and Figure 5b in the main text).

We also consider various ablations that gradually move the FIGNet model towards the DPI-Reimplemented particle-based model, as outlined in the ablations sections. Note that changing to a particle based representation makes it impossible to represent the floor using a dense representation, so all models with particles use an implicit floor (marked with * as in the main text).

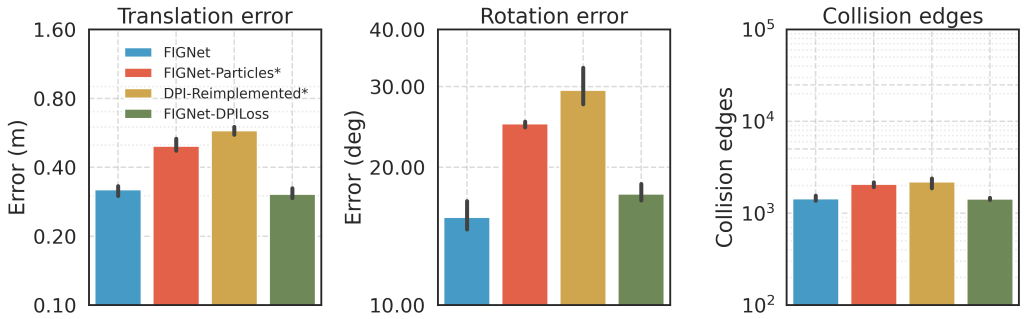


Figure E.4: Model ablations towards DPI on Movi-B dataset. While the loss formulation has little effect, replacing the mesh with densely sampled particles creates huge issues with memory requirements, and makes overall translation and rotation error worse. Models with * use an implicit floor representation.

F GENERALIZATION ACROSS SURFACE GEOMETRIES

To test generalization across different surface geometries, we created custom scenes where a sphere is dropped in some location and rolls along the different surfaces (Figure F.1). Despite never seeing inclined, curved or concave planes during training, FIGNet produces plausible trajectories of how the ball should roll. This supports our findings on the remarkable ability of FIGNet to generalize to new scenes, shapes and floor landscapes.

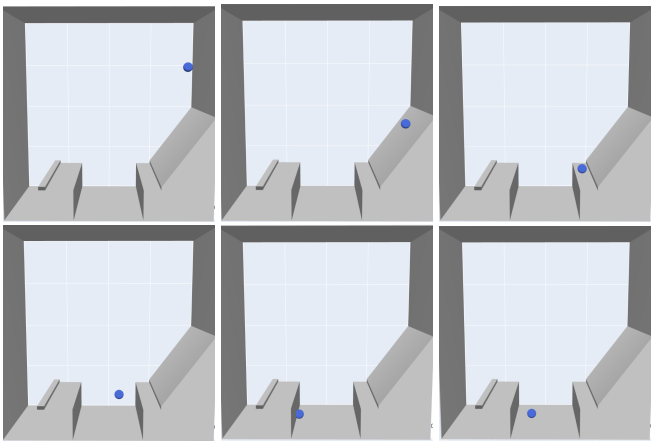


Figure F.1: Demonstration of generalization rollouts from Kubric MoviB to a 3D adaptation of the *Bridge* level from the Virtual Tools Game Allen et al. (2020). Grey and blue are static and dynamic objects respectively.

G GENERALIZATION ACROSS OBJECT SHAPES

To test generalization across different shapes, we train models on the MoviA or MoviB datasets, and test their generalization to MoviB or MoviC respectively, with no further fine-tuning. In Figure G.1a and Figure G.1b, FIGNet clearly outperforms alternative methods in generalization. Even more remarkably, FIGNet's performance on MoviB, when trained only on MoviA, still surpasses all baseline performances when the baselines are trained on MoviB directly. This suggests that FIGNet is not only accurate when trained, but provides compelling reasons to believe it will generalize to further complex dynamics in future.

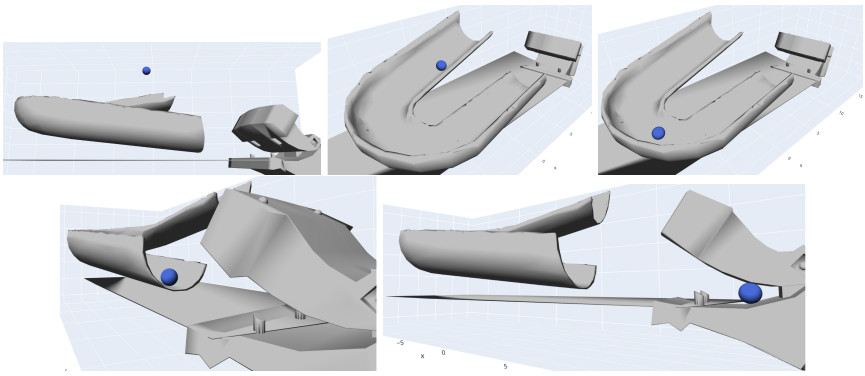


Figure F.2: Demonstration of generalization rollouts from Kubric MoviB to a manually designed u-slide and imported asset (`hippo.obj` asset imported from `github.com/mmacklin/tinsel.git`). Grey and blue are static and dynamic objects respectively.

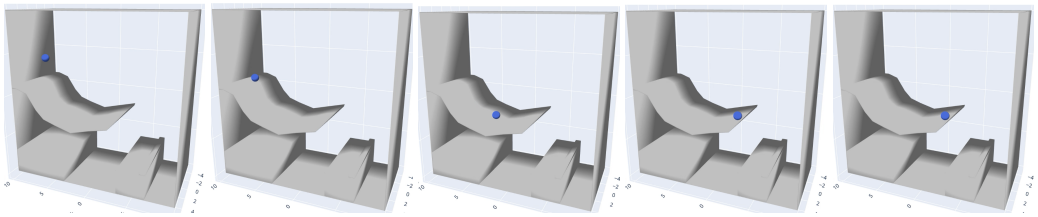
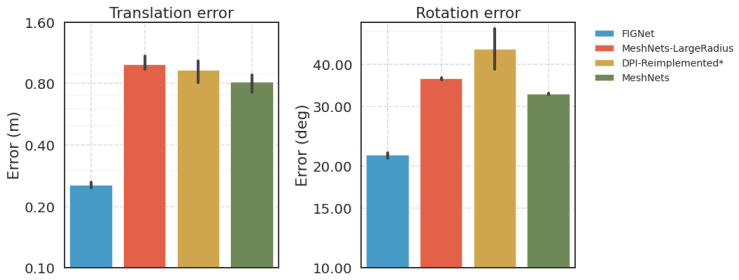
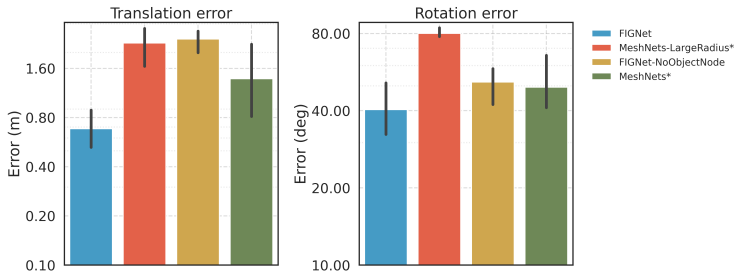


Figure F.3: Demonstration of generalization rollouts from Kubric MoviB to an unseen ramp.



(a) MoviA to MoviB



(b) MoviB to MoviC

Figure G.1: Generalization performance of the FIGNet model (a) trained on Kubric MoviA and tested on MoviB (b) trained on Kubric MoviB and tested on MoviC.

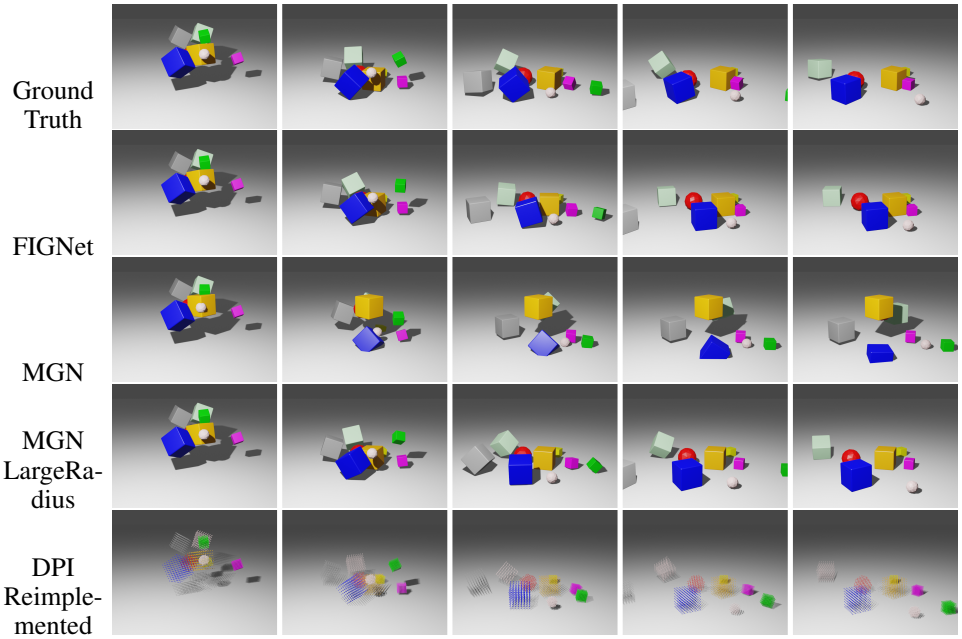


Figure G.2: Rollout examples on Movi-A dataset

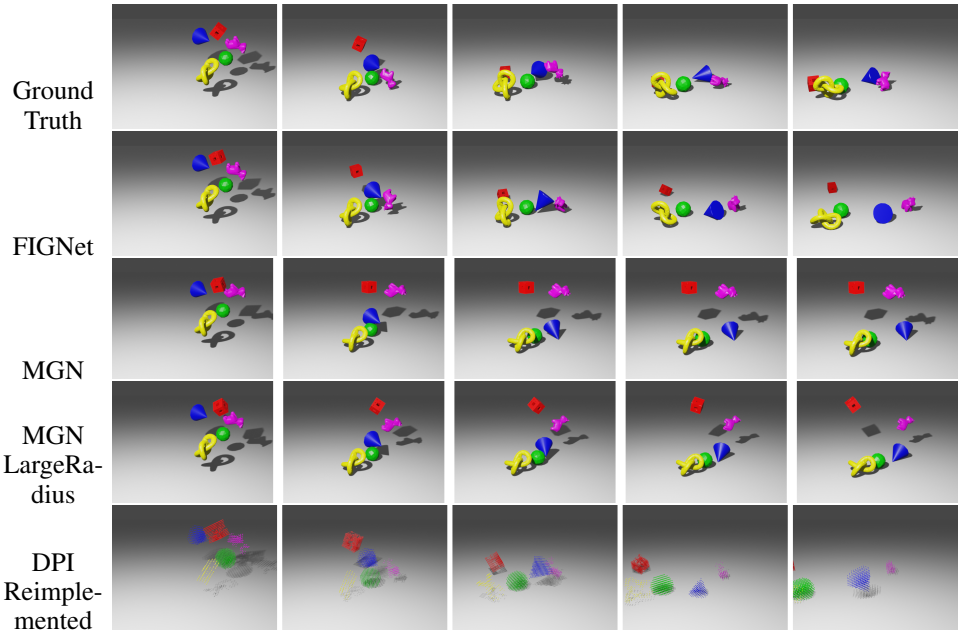


Figure G.3: Rollout examples on Movi-B dataset

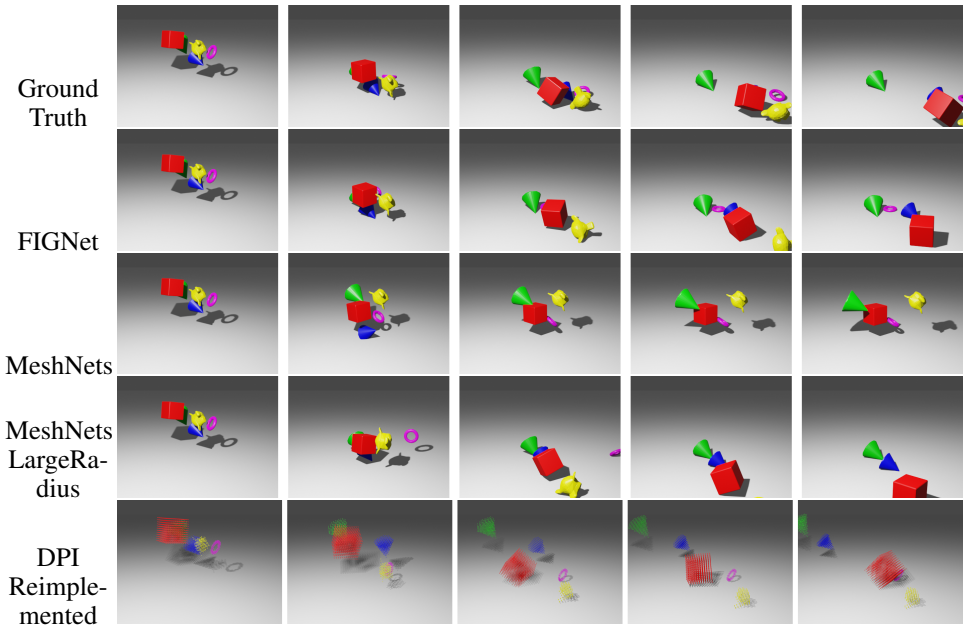


Figure G.4: Generalization from Movi-A to Movi-B dataset

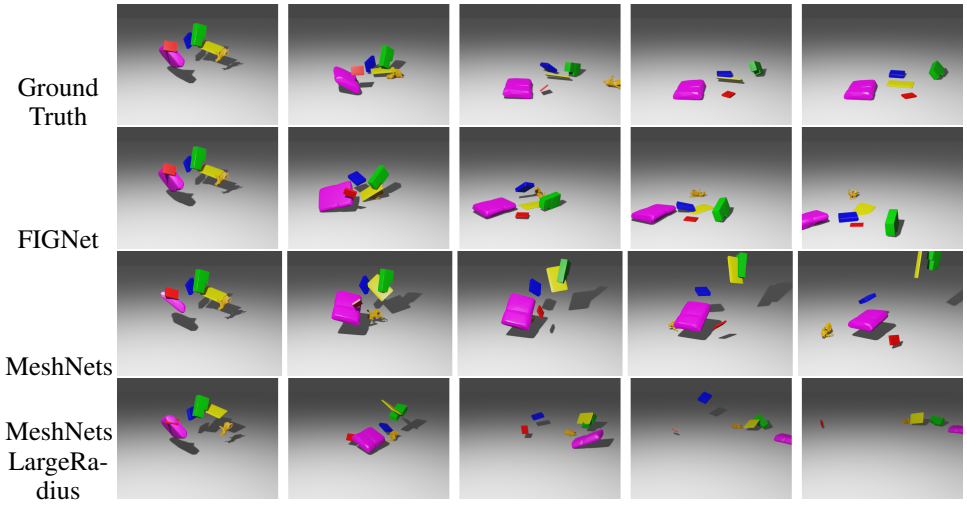


Figure G.5: Generalization from Movi-B to Movi-C dataset

H PENETRATION MEASURES

To measure penetration, we implement the algorithm from TODO. Since this algorithm is imperfect when faces are small, we report penetration statistics with respect to the ground truth trajectories given the same initial scene. FIGNet penetration distances are only 3% greater than the ground truth simulator, with only 4% more faces in penetration with each other relative to ground truth. By comparison, MeshGraphNets performs considerably worse when using a small collision radius, with penetration distances greater than 4x the ground truth model. These results are reported in Table H.1

Table H.1: Penetration distances and counts reported as a ratio to the distances and counts for the ground truth PyBullet simulator for the MoviA dataset.

Model	**Penetration distance ratio**	**Penetration count ratio**
FIGNet	1.037 ± 0.033	1.041 ± 0.037
MGN-LargeRadius+	1.071 ± 0.020	1.073 ± 0.018
MGN+	4.613 ± 0.143	5.246 ± 0.187

I RUNTIME

We additionally report runtime performance to more directly compare the amount of time taken to run a single forward step of each model. These results are reported in I.1 for the Movi-A dataset.

FIGNet has the shortest inference runtime in comparison to baselines. MGN-LargeRadius+, which is closest to FIGNet in terms of accuracy (Figure 5(a)), requires 2.7x more time to perform one inference step. DPI-Reimplemented* and MGN+ are inferior to FIGNet in terms of both runtime and accuracy.

Table I.1: Time of one inference step for each model on Movi-A dataset, in seconds.

Model	**Runtime (seconds)**
FIGNet	0.094 ± 0.005
MGN-LargeRadius+	0.258 ± 0.010
DPI-Reimplemented*	0.126 ± 0.006
MGN+	0.160 ± 0.007

J EFFECTS OF DIFFERENT COLLISION RADII

Figure J.1 demonstrates the ablations over different collision radii for the MeshGraphNets (MGN+) (Pfaff et al., 2021) model compared to FIGNet with collision radius 0.1 on the Kubric Movi-A dataset. Notably, there is no collision radius where MeshGraphNets would simultaneously have comparable runtime to FIGNet while also having comparable accuracy. Large values for the collision radius lead to better accuracy, but at the cost of runtime performance. The increase in collision radius also leads to an exponential increase in collision edges in MeshGraphNets, as more nodes fall into the collision radius, leading to higher memory consumption. In our experiments, the increase in collision edges manifested as 50% increase in runtime from collision radius 0.05 to 1.5.

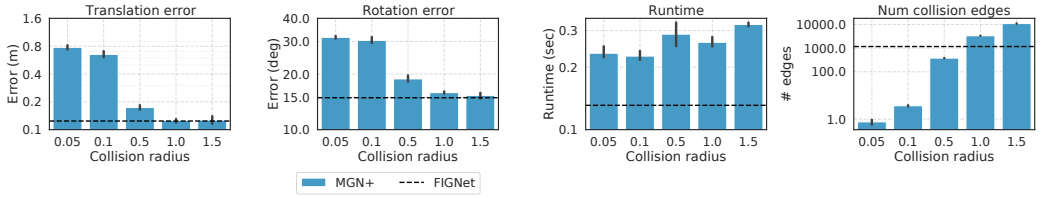


Figure J.1: Ablation over collision radii on Movi-A dataset for MGN, compared to FIGNet with collision radius 0.1. If the nodes (in MGN) or faces (in FIGNet) between different objects lie within the collision radius, we connect them with an edge in the graph.

By comparison FIGNet is not particularly sensitive to the collision radius. We trained models with a collision radius of 0.05 and 0.5 relative to the main model's radius of 0.1. The errors for these different collision radii are given in Table J.1

Table J.1: FIGNet collision radius sensitivity analysis for Movi-A dataset.

Metric	**Collision radius**		
	0.05	0.1	0.5
Translation error (m)	0.151 ± 0.014	0.115 ± 0.008	0.105 ± 0.006
Rotation error (deg)	16.8 ± 0.3	14.4 ± 0.2	13.4 ± 0.4

K RESULTS FROM MAIN TEXT AS TABLES

For ease of comparison between models, we provide the results on Kubric from the main text as tables.

Table K.1: Results on Kubric datasets from main text as table

Kubric Movi-A results			
Model	**Translation error**	**Rotation error**	**Collision edges**
DPI-Reimplemented*	0.180 ± 0.017	20.817 ± 0.926	2515.354 ± 34.341
MGN-LargeRadius+	0.119 ± 0.009	15.069 ± 0.649	10637.350 ± 317.336
MGN+	0.705 ± 0.069	31.210 ± 0.989	3.792 ± 0.396
FIGNet	0.115 ± 0.008	14.387 ± 0.175	232.644 ± 2.135
Kubric Movi-B results			
Model	**Translation error**	**Rotation error**	**Collision edges**
DPI-Reimplemented*	0.368 ± 0.057	26.928 ± 2.740	2250.688 ± 64.507
MGN-LargeRadius*	0.460 ± 0.045	26.342 ± 1.397	1797.985 ± 59.699
MGN*	0.538 ± 0.035	26.914 ± 0.783	34.367 ± 4.075
FIGNet	0.127 ± 0.006	13.990 ± 0.464	1385.610 ± 23.818