Contents

A VCD Implementation

A.1 GNN Architecture

As mentioned in the main paper, we take the network architecture in previous work [1] (referred to as GNS) for our dynamics GNN G_{dyn} and the edge GNN G_{edge}. Both GNN consists of three parts: encoder, processor and decoder. Since the dynamics and edge GNNs have very similar architectures, we first describe the architecture of the dynamics GNN, and then describe how the edge GNN architecture differs from that.

Input: The input to the dynamics GNN is a graph, where the nodes are the points in the voxelized point cloud of the cloth, and the edges consist of the collision edges (built using Eq. (1)) and mesh edges (inferred by a trained edge GNN). The node feature for a point v_i consists of the concatenation of its past m velocities, a one-hot encoding of the point type (picked or unpicked - see details about picking in Section 3.4 in the main paper and Section A.4 in the appendix), and the distance to the table plane. For edge e_{jk} that connects nodes v_j and v_k, its edge feature consists of the distance vector $(x_j - x_k)$, its norm $||x_j - x_k||$, a one-hot encoding of the edge type (mesh edge or collision edge), and the current displacement from the rest position $||x_j - x_k|| - r_{jk}$, where r_{jk} is the distance between x_j and x_k at the rest positions. The displacement from the rest positions are set to zero for collision edges which do not have rest positions.

We now describe how the robot action is incorporated into the input graph of the dynamics GNN as follows. As mentioned in Section 3.4 of the main paper, when we want to use the dynamics GNN to predict the effect of a pick-and-place robot action $a = \{a_{pick}, a_{place}\}$ on the current cloth, we first decompose the high-level action into a sequence of low-level movements, where each low-level movement is a small delta movement of the gripper and can be achieved in a short time. Specifically, we generate a sequence of small delta movements $\Delta x_1, ..., \Delta x_H$ from the high-level action, where $x_{pick} + \sum_{i=1}^{H} \Delta x_i = x_{place}$. Each delta movement Δx_i moves the gripper a small distance along the pick-and-place direction and the motion can be predicted by the dynamics GNN in a single step. We then incorporate the small delta movement into the input graph as follows. When the gripper is grasping the cloth, we denote the picked point as u. We assume that the picked point is rigidly attached to the gripper; thus, when considering the effect of the t^{th} low-level movement of the robot gripper, we modify the input graph by directly setting the picked point u's position $x_{u,t} = x_{pick} + \sum_{i=1}^{t} \Delta x_i$ and velocity $\dot{x}_{u,t} = \Delta x_i / \Delta t$, where Δt is the time for one low-level movement step. The dynamics GNN will then propagate the effect of the robot action along the graph when predicting future states.

Encoder: The encoder consists of two separate multi-layer perceptrons (MLP), denoted as ϕ_p, ϕ_e, that map the node and edge feature, respectively, into latent embedding. Specifically, the node encoder ϕ_p maps the node feature for node v_i into the node embedding h_i, and the edge encoder ϕ_e maps the edge feature for edge e_{jk} into the edge embedding g_{jk}.

Processor: The processor consists of L stacked Graph Network (GN) blocks [2] that update the node and edge embedding, with residual connections between blocks. We use $L = 10$ in both edge GNN G_{edge} and dynamics GNN G_{dyn}. The l^{th} GN block contains an edge update MLP f_e^l and a node update MLP f_p^l that take as input the edge and node embedding g^l and h^l respectively and outputs updated embedding g^{l+1} and h^{l+1} (we denote g^0 and h^0 as the edge and node embedding output by the encoder). It also contains a global update MLP f_c^l that takes as input a global vector embedding c^l, and outputs the updated global embedding c^{l+1}. The initial global embedding c^0 is set to be 0. For each GN block, first the edge update MLP updates the edge embedding; it takes as input the current edge embedding g_{jk}^l, the node embedding h_j^l, h_k^l for the nodes that it connects, as well as the global embedding c^l: $g_{jk}^{l+1} = f_e^l(h_j^l, h_k^l, g_{jk}^l, c^l) + g_{jk}^l, \quad \forall e_{jk} \in E$. The node update MLP then updates the node embedding; its input consists of the current node embedding h_i^l, the sum of the updated edge embedding for the edges that connect to the node, and the global embedding c^l: $h_i^{l+1} = f_p^l(h_i^l, \sum_j g_{ji}^{l+1}, c^l) + h_i^l, \quad \forall i = 1, ..., N_p$. Note the edge and node updates both have residual connections between consecutive blocks. Finally, the global update MLP takes as input the current global embedding c^l, the mean of the updated node and edge embedding, and updates the global embedding as: $c^{l+1} = f_c^l(c^l, \frac{1}{|V|}\sum_{i=1}^{|V|} h_i^{l+1}, \frac{1}{|E|}\sum_{e_{jk}} g_{jk}^{l+1})$.

Decoder: The decoder is an MLP ψ that takes as input the final node embedding h_i^L output by the processor for each point v_i; the decoder outputs the acceleration for each point: $\ddot{x}_i = \psi(h_i^L)$. The acceleration can then be integrated using the Euler method to update the node position x_i. We train the graph GNN G_{dyn} using the L2 loss between the predicted point acceleration $\ddot{x}_i$ and the ground-truth acceleration obtained by the simulator; see Sec. A.2 for details.

Edge GNN: The edge GNN G_{edge} has nearly the same architecture as the dynamics GNN, with the following differences: first, the input graph to the edge GNN encoder consists of only the voxelized point cloud and the collision edges $\langle P, E^C \rangle$; the edge GNN aims to infer which collision edges are also mesh edges. The node feature is 0 for all nodes. The edge feature for edge e_{jk} consists of the distance vector $(x_j - x_k)$ and its norm $||x_j - x_k||$ (without the edge type, since this must be inferred by the edge GNN). The processor is exactly the same as that in the dynamics GNN. The decoder is an MLP that takes as input the final edge embedding output by the processor and outputs the probability of the collision edge being a mesh edge. We use a binary classification loss on the prediction of the mesh edge for training.

Hyperparameters In simulator, we set the radius of particles to be 0.00625, an All MLPs that we use has three hidden layers with 128 neurons each and use ReLU as the activation function. The detailed parameters of the GNN architecture, as well as the simulator parameters, can be found in Supplementary Table 1.

A.2 VCD Training Details

Details about training in simulation: We trian the dynamics GNN with one-step prediction loss: suppose that we sample a transition (V_t, a_t, V_{t+1}), where a_t is a low-level action. Then we assign the velocity at timestep t that is input to the network to be the ground-truth velocity obtained from the simulator (after matching the points to their corresponding simulator particles). This strategy enables us to sample arbitrary timesteps for training rather than needing to always simulate the dynamics from the first timestep.

For training the edge GNN, we need to obtain the ground-truth of which collision edges are also mesh edges. During simulation training, a collision edge is assumed to be a mesh edge if the mapped simulation particles of the edge's both end points are connected by a spring in the simulator.

We train our dynamics GNN with the ground-truth mesh edges, and directly use it with the mesh edges predicted by the edge GNN at test time. We find this to work well without fine-tuning the dynamics GNN on mesh edges predicted by the edge GNN, due to the high prediction accuracy(91%) of the edge GNN.

Details about bipartite graph matching: As mentioned in the main paper, we need bi-partite graph matching to find a mapping from the voxelized point cloud to the simulation particles, in order to obtain the state and connectivity of the voxelized point cloud for training the dynamics and edge GNN. Given N points in the voxelized point cloud $p_i, i = 1 \dots N$ and M simulated particles of the cloth in simulation $x_j, j = 1 \dots M$, the goal of the bipartite graph matching here is to match each point in the point cloud to a simulated particles. The simulated cloth mesh is downsampled by three times to improve computation efficiency, e.g., a cloth composed of 40×40 particles is downsampled to be of size 13×13. The bi-partite matching is only performed on the downsampled particles. We build the bipartite graph by connecting an edge from each p_i to x_j, with the cost of the edge being the distance between the two points. In our experiments, we always have $M > N$ since we use a large grid size for the voxelization.

Training data: We collect 2000 trajectories, each consisting of 1 pick-and-place action. The pick point is randomly chosen among the locally highest points on the cloth; this is only done to generate the training data for the dynamics model, not for planning (we do this for training the VSF and CFM baselines as well; the MVP baseline uses the behavioral policy to generate its training data). The unnormalized direction vector $p = (\Delta x, \Delta y, \Delta z)$ for the pick-and-place action is uniformly sampled as follows: $\Delta x, \Delta z \in [-0.5, 0.5]$, $\Delta y \in [0, 0.5]$. The direction vector is then normalized and the move distance is sampled uniformly from $[0.15, 0.4]$. The high-level pick-and-place action is decomposed into 100 low-level steps: the pick-and-place is executed in the 60 low-level actions, and then we wait 40 steps for the cloth to stablize. We train our dynamics model in terms of low-level actions.

We choose the voxel size (0.0216) to be three times of the particle radius (0.00625) to keep it consistent with the downsampled mesh. The neighbor radius, which determines the construction of collision edges, is set to be roughly two times of the voxel size, so as to ensure that particles in adjacent voxels are connected.

Training parameters: We use Adam [3] with an initial learning rate of 0.0001 and reduce it by a factor of 0.8 if the training plateaus. We train with a batch size of 16. The training of the dynamics GNN takes roughly 4 days to converge on a RTX 2080 Ti. The training of the edge GNN usually converges in 1 or 2 days. Detailed training parameters can be found in Supplementary Table 1.

Model parameter	Value
Encoder(same for both node encoder and edge encoder)	
number of hidden layers	3
size of hidden layers	128
Processor	
number of message passing steps	10
number of hidden layers in each edge/node update MLP	3
size of hidden layers	128
Decoder	
number of hidden layers	3
size of hidden layers	128
Training parameters	**Value**
learning rate	0.0001
batch size	16
training epoch	120
optimizer	Adam
beta1	0.9
beta2	0.999
weight decay	0
Others	**Value**
dt	0.05 second
particle radius	0.00625 m
downsample scale	3
voxel size	0.0216 m
neighbor radius R	0.045 m

Supplementary Table 1: Summary of all hyper-parameters.

A.3 Graph-imitation Training Details

Although VCD performs decently under partial observability, we found dynamics model trained on full mesh model usually converges faster and obtains better asymptotic performance. This is well expected since incomplete information caused by self-occlusion results in ambiguity of state estimation.

Therefore, we introduce graph-imitation learning to inject prior knowledge of the full cloth into the dynamics model. The prior encodes structure of the full cloth and incentivizes the model to reason about occlusion implicitly.

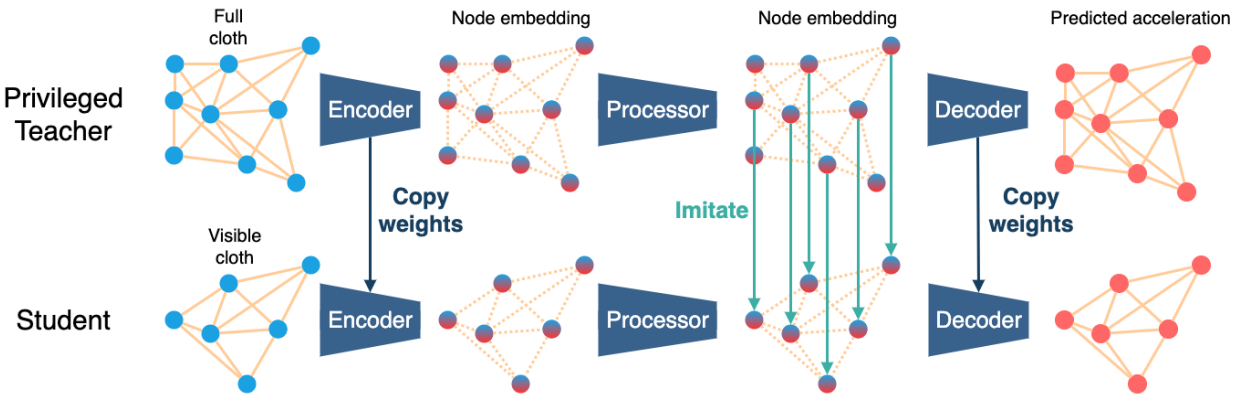
A.3.1 Privileged Graph-imitation Learning

The main spirit of privileged graph-imitation learning is to train a student model which takes partial point cloud as input, to imitate a privileged agent which has access to privileged information. We hope the student to learn a recover function that recovers true states from partial information. A visual illustration is shown in Supplementary Figure 1.

To do so, we first train a privileged agent with all simulated particles(including the occluded ones) and ground-truth mesh edges. The privileged teacher model shares identical architecture as the student model, but with complete information. We train the teacher model with acceleration loss and the auxiliary reward prediction loss.

Graph-based imitation learning is not straightforward because the graphs of two models have very different structures. Typically, the graph of teacher model will have more vertices since it can observe occluded particles while the student only observes the voxelized partial point cloud. To tackle with this challenge, we conduct bipartite matching to match student nodes with teacher nodes as described in A.2.

Once we have the node correspondence, we retrieve the intermediate node features of both teacher and



Supplementary Figure 1: A graphical illustration of privileged graph imitation learning. The privileged teacher has the same model architecture as student, but takes full cloth as input. Following [4], we initialize the encoder and decoder of student model by weights of pretrained teacher. Then we freeze the teacher and transfer the privileged information by matching the node embedding and global embedding of two models. The target nodes to imitate are obtained by bi-partite matching as described in A.2.

student model, h_T^L and h_S^L, and force the node feature of student h_S^L to be similar to h_T^L. The final output is still supervised by groundtruth acceleration. We copy the weights of encoder and decoder from teacher model to initialize student since we find it accelerate training. The teacher is frozen during imitation learning. By imitating the intermediate node features, we provide high capacity training signal to the student to recover groundtruth acceleration by proper message passing. To successfully imitate the teacher, the student have to conduct occlusion reasoning to some extent, and take the effects of occluded particles and erroneous mesh edges into account. In addition to node features, the student model also mimic the global embedding of teacher model to make more accurate reward prediction. We use mean square error for imitation learning.

A.3.2 Auxiliary Reward Prediction

Following [5], we additionally train our dynamics model to predict reward in order to regularize the model. The groundtruth reward, which is the coverage of cloth after one time step, is calculated by approximation as described in A.4. The coverage is calculated over all particles, thus it provides information from a global view to the model. The reward model is a three-layer MLP which takes global embedding c^L as input. We use mean square error to train the model.

It should be noted that at test time, we still use the heuristic reward function, which models particles as spheres and calculate an approximate coverage on the partial point cloud. Although the learned reward model predicts a global reward, which theoretically will take into the newly revealed occluded regions into account, we found it perform slightly worse than the heuristic reward function.

A.4 VCD Planning Details

We summarize the planning procedure of VCD in Algorithm 1. We sample K high-level pick-and-place actions. For each sampled high-level action, we roll out our dynamics model using that action for H low-level steps and obtain the sequence of predicted point positions.

Action sampling during planning in simulation As described in the main text, we sample 500 pick-and-place actions, where the pick point is first uniformly sampled from a bounding box of the cloth and then projected to be on the cloth mask. For generating the bounding box, we first obtain the cloth mask from the simulator. We then obtain the minimal and maximal pixel coordinates u, v value of the cloth mask. The bounding box is the rectangle with corners $(min(u) - padding, min(v) - padding)$ and $(max(u) + padding, max(v) + padding)$, where padding is set to be 30 pixels for the 360×360 image size we use. We use rejection sampling to make sure the place point is within the image to keep the action within

Algorithm 1: Planning with pick-and-place actions

input : Voxelized partial point cloud P, Edge GNN G_{edge}, Dynamics GNN G_{dyn}, number of sampled actions K

output: pick-and-place action $a = \{x_{pick}, x_{place}\}$

Build collision edges E_0^C with P; Infer mesh edges $E^M \leftarrow G_{edge}(P, E_0^C)$

for $i \leftarrow 1$ ***to*** K **do**

  Sample a pick-and-place action x_{pick}, x_{place}

  Compute low-level actions $\Delta x_1, ..., \Delta x_H$

  Get picked point v_u from x_{pick}

  Pad historic velocities with 0: $\mathbf{x}_0 \leftarrow P, \dot{\mathbf{x}}_{-m...0} \leftarrow \mathbf{0}$

  for $t \leftarrow 0$ ***to*** H **do**

    Build collision edges E_t^C with $\mathbf{x}_t$

    Move picked point according to gripper movement by :

    $x_{u,t} \leftarrow x_{u,t} + \Delta x_t, \;\; \dot{x}_{u,t} \leftarrow \Delta x_t / \Delta t$

    Predict accelerations using G_{dyn}: $\ddot{\mathbf{x}}_t \leftarrow G_{dyn}(\mathbf{x}_t, \dot{\mathbf{x}}_{t-m...t}, u, E^M, E_t^C)$

    Update point cloud predicted positions & velocities:

    $\dot{\mathbf{x}}_{t+1} = \dot{\mathbf{x}}_t + \ddot{\mathbf{x}}_t \Delta t, \;\; \mathbf{x}_{t+1} = \mathbf{x}_t + \dot{\mathbf{x}}_{t+1} \Delta t$

    Readjust picked point according to gripper movement by

    $x_{u,t} \leftarrow x_{u,t} + \Delta x_t, \;\; \dot{x}_{u,t} \leftarrow \Delta x_t / \Delta t$

  end

  Compute reward r based on final point cloud predicted position $\mathbf{x}_H$

end

return *pick and place action with maximal reward*

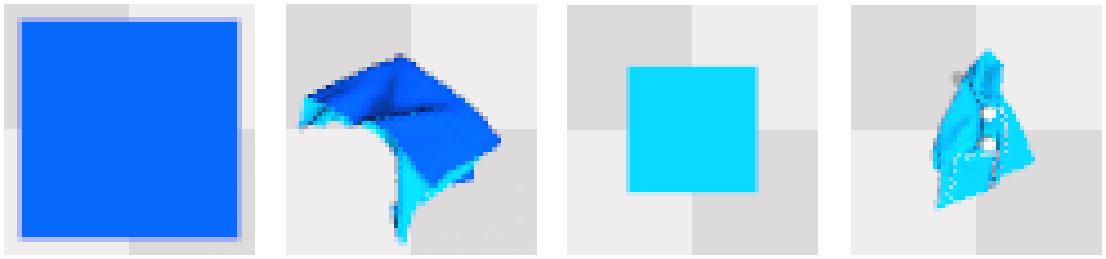
the depth camera view. The unnormalized direction vector $p = (\Delta x, \Delta y, \Delta z)$ (y is the up axis) of the pick-and-place is uniformly sampled as follows: $\Delta x, \Delta z \in [-0.5, 0.5]$, and $\Delta y \in [0, 0.5]$. The vector is normalized and then the distance is separately sampled from $[0.05, 0.2]$ meters. We decompose the pick-and-place action into 10 low-level actions and wait for another 6 steps for the cloth to stabilize.

Action sampling during planning in the real world The robot action space is pick-and-place with a top down pinch grasp. For each action, we sample 100 pick-and-place actions to be evaluated by our model. Each action sample is generated as follows: We first sample a pick-point location corresponding to the segmented cloth, denoted as (p_x, p_y). We then generate a random direction $\theta \in [0, 2\pi]$ and distance $l \in [0.02, 0.1]$ meters. Then the place point will be $(p_x + l\cos\theta), p_y + l\sin\theta$. We only accept an action if both the pick and the place points are within the work space of the robot. We additionally filter out actions whose place points are overlapping with the cloth. This heuristic saves computation time without sacrificing performance.

Reward computation in planning: As described in the main text, to compute the reward function r for planning, we treat each node in the graph as a sphere with radius R and compute the covered area of these spheres when projected onto the ground plane. To prevent the planner from exploiting the model inaccuracies, we do the following: if the model predicts that there are still points above a certain height threshold after executing the pick-and-place action and waiting the cloth to stabilize, then the model must be predicting inaccurately and we set the reward of such actions to 0. The threshold we use is computed as 15×0.00625 meters, where 0.00625 is the radius of the cloth particle used in the simulation.

B Baselines Implementation

For all the baselines, we try our best to adjust the SoftGym cloth environment to match the cloth environment used in the original papers. For VSF, we place the camera to be top-down and zoomed in so that the cloth covers the entire image when fully flattened. We also changed the color of the cloth to be bluish as in the original paper. We collect 7115 trajectories, each consisting of 15 pick-and-place actions for training the VSF model (same as in the VSF paper). For CFM, we also use a top-down camera and change the color of



(a) VSF goal image. (b) VSF observation. (c) CFM goal image. (d) CFM observation.

Supplementary Figure 2: Images of cloth configurations used by the baseline methods.

the cloth to be the same on both sides, following the suggestion of the authors (personal communication). We collect 8000 trajectories each consisting of 50 pick-and-place actions for training the contrastive forward model (same as in the CFM paper). For MVP, we collect 5000 trajectories each with 50 pick-and-place actions and report the performance of the best performing model during training. We trained each of the baselines for at least as many pick-and-place actions as they were trained in their original papers. For training our method, VCD, we collect 2000 trajectories, each consisting of 1 pick-and-place action decomposed into 20 low-level actions for training. Note that this is fewer pick-and-place actions than any of the baselines used for training. We now describe each compared baseline in more details below:

B.1 VisuoSpatial Forsight (VSF)

We use the official code of VSF provided by the authors[1].

Image: Following the original paper, we use images of size 56×56. we place the camera to be top-down and zoomed in so that the cloth covers the entire image when fully flattened. An example goal image of the smoothed cloth for VSF is shown in Supplementary Figure 2.

Training data & Procedure: For training the VSF model, we collect 7115 trajectories for training (same as in the VSF paper), each consisting of 15 pick-and-place actions. Following the VSF paper, the pick-and-place action first moves the cloth up to a fixed height, which is set to be 0.02 m in our case, and then moves horizontally. The horizontal movement vector is sampled from $[-0.07, 0.07] \times [-0.07, 0.07]$ m. This range is smaller than what is used for VCD, as we follow the original paper to set the maximal move distance roughly half of the cloth/workspace size. We use rejection sampling to ensure the after the movement, the place point is within the camera view. Similar to VCD, the pick point is uniformly sampled among the locally highest points on cloth (only during training). It takes 2 weeks for VSF to converge on this dataset.

Action sampling during planning: Similarly to VCD, the pick point is sampled uniformly from a bounding box around the cloth and then projected to the cloth mask. The padding for the bounding box here we use is 6. Other than the pick point, other elements of the pick-and-place action is sampled following the exact same distribution as in the training data collection.

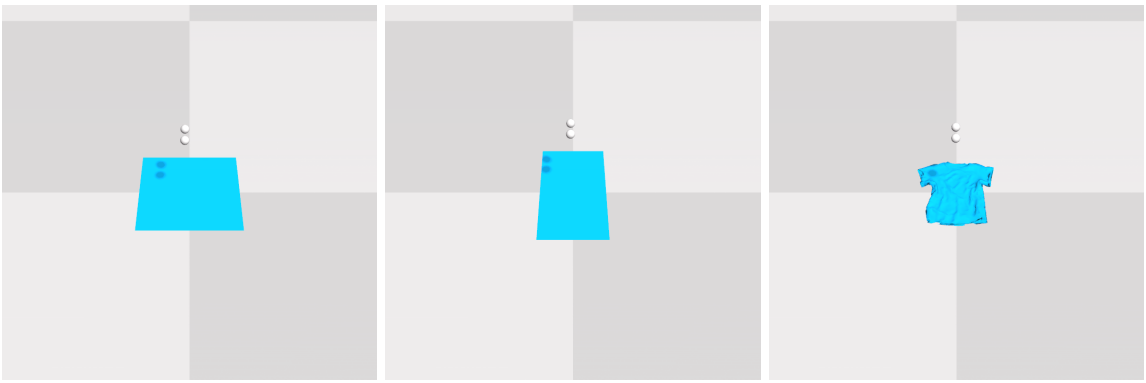
B.2 Contrastive Forward Model (CFM)

We use the official code of CFM provided by the authors[2].

Image: Following the original paper, we use images of size 64×64. We also place the camera to be top-down and adjust the camera height so the cloth contains a similar portion of the image as in the original paper. Following the suggestions from the authors (personal communication), we also set the color of the cloth to be the same on both sides. See Supplementary Figure 2 for an example of the images we use.

Training data: For training, we collect 8000 trajectories each consisting of 50 pick-and-place actions, which is the same as in the original paper. Similar to VCD and VSF, the pick point is sampled among the locally highest points on the cloth (only during training). The movement vector is sampled from

[1] https://github.com/ryanhoque/fabric-vsf
[2] https://github.com/wilson1yan/contrastive-forward-model



Supplementary Figure 3: Images of the square cloth, rectangular cloth, and t-shirt used in simulation.

$[-0.04, 0.04] \times [0, 0.04] \times [-0.04, 0.04]$ m, where the y-axis is the negative gravity direction. We use pick-and-place actions with such small distances following the original paper. We also use rejection sampling to ensure the place point is within the camera view.

Action sampling during planning: Similar to VCD, the pick point is sampled uniformly from a bounding box around the cloth and then projected to the cloth mask. The padding size here we use for the bounding box is 5. Other than the pick point, other elements of the pick-and-place action are sampled following the exact same distribution as in training data collection.

B.3 Maximal Value under Placing (MVP)

We use the official code of MVP provided by the authors[3].

Image: Following the original paper, we use images of size 64×64. We also place the camera to be top-down.

Training data: For training, we collect 8000 trajectories each consisting of 50 pick-and-place actions, which is the same as the original paper. However, the Q function starts to diverge after 5000 trajectories and the performance starts to drop. Thus we report the best policy performance when it has been trained for 5000 trajectories. This corresponds to around 15000 training iterations.

Action space: The action space for the MVP policy is in 5 dimension: $(u, v, \Delta x, \Delta y, \Delta z)$, where u, v is the image coordinate of the pick point and is sampled for the segmented cloth pixel. We use the depth information to back project the pick point to 3d space. $(\Delta x, \Delta y, \Delta z)$ is the displacement of the place location relative to the pick point and is clipped to be within 0.5. Additionally, the height Δy is clipped to be non-negative.

C Experimental Setup

C.1 Simulation Setup

We use the Nvidia Flex simulator, wrapped in SoftGym [6], for training. In SoftGym, the robot gripper is modeled using a spherical picker that can move freely in 3D space and can be activated so the nearest particle will be attached to it. For the simulation experiments, we use a nearly square cloth, composed of a variable number of particles sampled from $[40, 45] \times [40, 45]$; this corresponds to a cloth of size in the range of $[25, 28] \times [25, 28]$ cm. Detailed cloth parameters such as stiffness are listed in the appendix. For all methods, we randomly generate 20 initial cloth configurations for training. The initial configurations are generated by picking the cloth up and then dropping it on the table in simulation. For evaluation, we consider three different geometries: 1) the same type of square cloth as used in training; 2) Rectangular cloth. The length and width of the rectangular cloth is sampled from $[19, 21] \times [31, 34]$ cm. 3) T-shirt. Images of these three shapes of cloth in simulation are shown in Supplementary Figure 3.

[3] https://github.com/wilson1yan/rlpyt

We set the stiffness of the stretch, bend, and shear spring connections to $0.8, 1, 0.9$, respectively.

C.2 Real-world Setup

Real World Setup Our real robot experiments use a Franka Emika Panda robot arm with a standard panda gripper. We obtain RGBD from a side view Azure Kinect camera and crop the RGBD image into the size of [345, 425], which corresponds to a workspace of 0.4 x 0.5 meters. To obtain the cloth point cloud, we first use color thresholding to remove the table background and obtain the cloth segmentation mask and then back project each cloth pixel to 3d space using the depth information. We evaluate on three pieces of cloth: Two squared towels made of silk and cotton respectively and one shirt made of cotton. We use the covered area as described in Sec. A.4 as our reward function.

For each cloth, we evaluate 12 trajectories each with a maximum of 20 pick-and-place actions. For each trajectory, the robot stops if the normalized performance is higher than 0.95 or if the predicted rewards of all the sampled actions are smaller than the current reward. For each trajectory, we reset the cloth configuration using the following protocol: Each time, the arm picks a random point on the cloth, lifts it up to 0.4 meters above the table and drop it at a fixed point on the table. This procedure is done three times in the beginning of each trajectory.

D Additional Experimental Results

D.1 Simulation Experiments

D.1.1 Normalized Improvement and Normalized Coverage in Simulation

NI and NC of our simulation experiments are reported in Supplementary Table 2 and Supplementary Table 3. With different metrics, our method consistently outperforms all baselines.

	# of pick-and-place actions / Method	5	10	20	50
Square	VCD (Ours)	0.624 ± 0.217	0.778 ± 0.222	0.968 ± 0.307	1.000 ± 0.043
	VCD-graph-imitation (Ours)	$\mathbf{0.692 \pm 0.258}$	$\mathbf{0.919 \pm 0.377}$	$\mathbf{0.990 \pm 0.122}$	$\mathbf{1.000 \pm 0.039}$
	VSF [7]	0.321 ± 0.112	0.561 ± 0.127	0.767 ± 0.134	0.968 ± 0.021
	CFM [8]	0.053 ± 0.051	0.077 ± 0.053	0.109 ± 0.066	0.105 ± 0.106
	MVP [9]	0.399 ± 0.210	0.435 ± 0.137	0.421 ± 0.361	0.307 ± 0.310
Rectangular	VCD (Ours)	$\mathbf{0.585 \pm 0.359}$	$\mathbf{0.918 \pm 0.413}$	0.973 ± 0.341	0.979 ± 0.399
	VCD-graph-imitation (Ours)	0.556 ± 0.372	0.912 ± 0.393	$\mathbf{0.985 \pm 0.164}$	$\mathbf{1.000 \pm 0.028}$
	VSF [7]	0.268 ± 0.090	0.356 ± 0.163	0.542 ± 0.177	0.715 ± 0.162
T-shirt	VCD (Ours)	$\mathbf{0.595 \pm 0.279}$	0.533 ± 0.285	0.738 ± 0.465	$\mathbf{0.979 \pm 0.399}$
	VCD-graph-imitation (Ours)	0.595 ± 0.385	$\mathbf{0.633 \pm 0.357}$	$\mathbf{0.838 \pm 0.450}$	0.969 ± 0.8860
	VSF [7]	-0.009 ± 0.125	0.004 ± 0.188	0.176 ± 0.237	0.219 ± 0.218

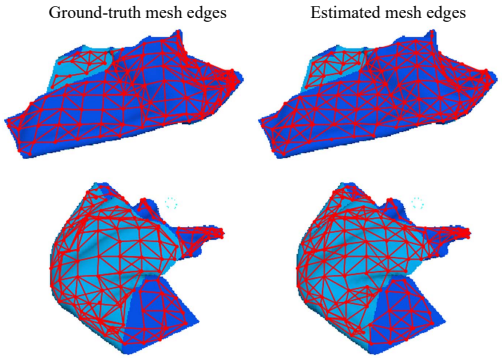
Supplementary Table 2: Normalized Improvement (NI) of all methods in simulation, for varying numbers of allowed pick and place actions.

D.1.2 Visualization of Edge GNN

We compare predictions of our edge prediction model with the ground-truth edges used for training the edge model in Supplementary Figure 4. As shown, the edge GNN prediction reasonably matches the ground-truth, and thus well captures the cloth structure; it can also correctly disconnect the top layer from the bottom layer when the cloth is folded, e.g., the top left part of the first example and the bottom right part of the second example. Note our method uses only the point cloud as input and the color in this figure is only used for visualization. The edge GNN is trained on the same dataset as the dynamics GNN (described in Sec. A.2), and on the validation set, it achieves a prediction accuracy of 0.91.

	Method \ # of pick-and-place actions	5	10	20	50
Square	VCD (Ours)	0.776 ± 0.132	0.872 ± 0.128	0.985 ± 0.1873	1.000 ± 0.023
	VCD-graph-imitation (Ours)	**0.837 ± 0.150**	**0.966 ± 0.236**	**0.994 ± 0.076**	**1.000 ± 0.021**
	VSF [7]	0.629 ± 0.053	0.762 ± 0.093	0.878 ± 0.090	0.984 ± 0.010
	CFM [8]	0.445 ± 0.052	0.466 ± 0.044	0.494 ± 0.031	0.538 ± 0.044
	MVP [9]	0.667 ± 0.121	0.667 ± 0.124	0.661 ± 0.194	0.609 ± 0.179
Rectangular	VCD (Ours)	**0.785 ± 0.182**	**0.957 ± 0.233**	0.985 ± 0.183	0.998 ± 0.017
	VCD-graph-imitation (Ours)	0.768 ± 0.191	0.949 ± 0.215	**0.992 ± 0.080**	**1.000 ± 0.015**
	VSF [7]	0.622 ± 0.078	0.664 ± 0.078	0.765 ± 0.119	0.860 ± 0.072
T-shirt	VCD (Ours)	0.837 ± 0.107	0.828 ± 0.096	0.897 ± 0.150	**0.991 ± 0.189**
	VCD-graph-imitation (Ours)	**0.867 ± 0.143**	**0.901 ± 0.179**	**0.960 ± 0.218**	0.991 ± 0.331
	VSF [7]	0.636 ± 0.086	0.653 ± 0.090	0.676 ± 0.079	0.698 ± 0.075

Supplementary Table 3: Normalized coverage (NC) of all methods in simulation on the regular cloth, for varying numbers of allowed pick and place actions.



Supplementary Figure 4: The edge prediction result of our edge GNN. Red lines visualize the ground-truth (left) or inferred (right) mesh connections.

D.1.3 Visualizations of Planned Actions in Simulation

Supplementary Figure 5 shows three planned pick-and-place action sequences of VCD in simulation. As shown, VCD successfully plans actions that gradually smooths the cloth. We observe note that VCD favours picking edge / corner points and pulling outwards, which is an effective smoothing strategy, demonstrating the effectiveness of VCD for planning.

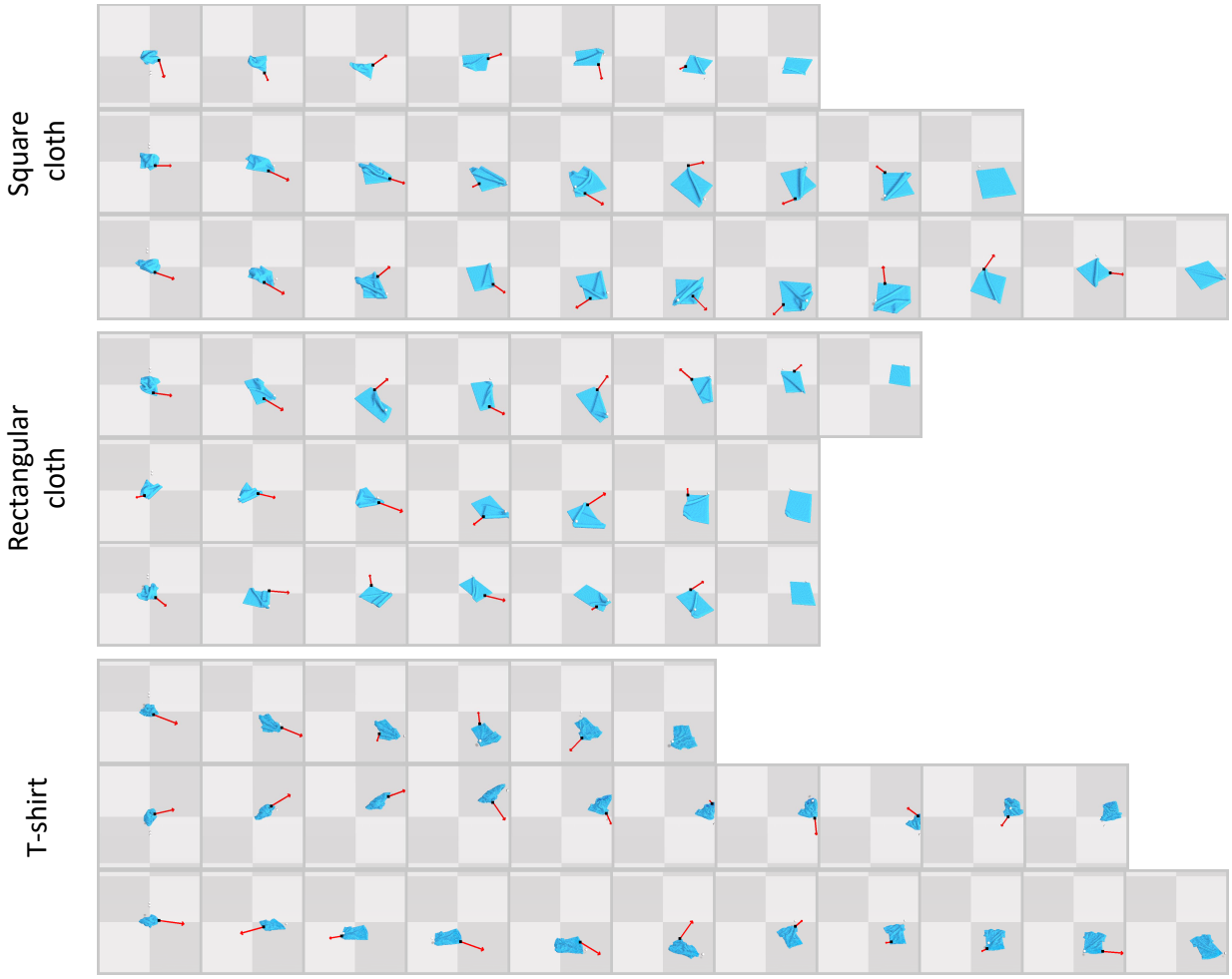
D.1.4 Visualizations of Open-loop Predictions in Simulation

In order to understand better what our model is learning, we visualize the prediction of our model compared to the simulator output in Supplementary Figure 6, 7, 8. Given a pick-and-place action decomposed into 75 low-level actions, the model is given the 5^{th} point cloud in the trajectory with the past 4 historical velocities, and the dynamics model is used to generate the future predictions. As shown, even if the prediction horizon is as long as 70 steps, VCD is able to give relatively accurate predictions on all cloth shapes, indicating the effectiveness of incorporating the inductive bias of the cloth structure into the dynamics model.

D.2 Robot Experiments

D.2.1 Running Time

In average, it takes 12.7 seconds for VCD to plan each pick-and-place action (100 samples) on 4 RTX 2080Ti and 10.2 seconds for Franka to execute the action. With additional communication overhead, our current system takes around 40 seconds for computing and executing each pick-and-place action.



Supplementary Figure 5: Three example planned pick-and-place action sequences for square cloth, rectangular cloth, and t-shirt. All trajectories shown achieve a normalized improvement above 0.98.

D.2.2 Normalized Coverage (NC) of Robot Experiments

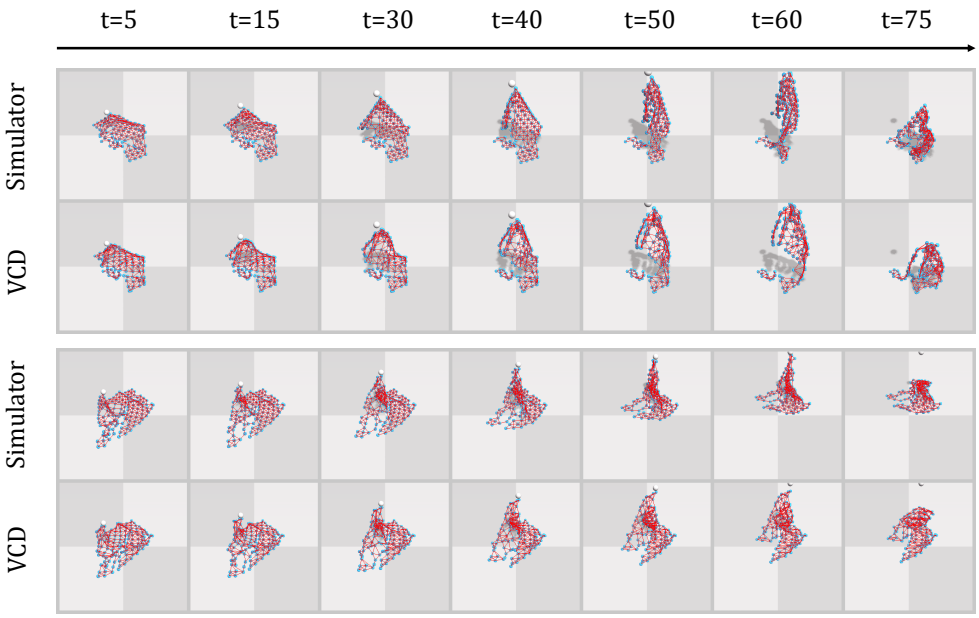
For the robot experiments, the main text reports the normalized improvement (NI). NC are reported here in Supplementary Table 4.

Material \ # of pick-and-place actions	5	10	20	Best
Cotton Square Cloth	0.690 ± 0.166	0.884 ± 0.293	0.959 ± 0.193	0.959 ± 0.080
Silk Square Cloth	0.744 ± 0.180	0.876 ± 0.314	0.964 ± 0.075	0.964 ± 0.054
Cotton T-Shirt	0.548 ± 0.114	0.601 ± 0.093	0.688 ± 0.068	0.773 ± 0.141

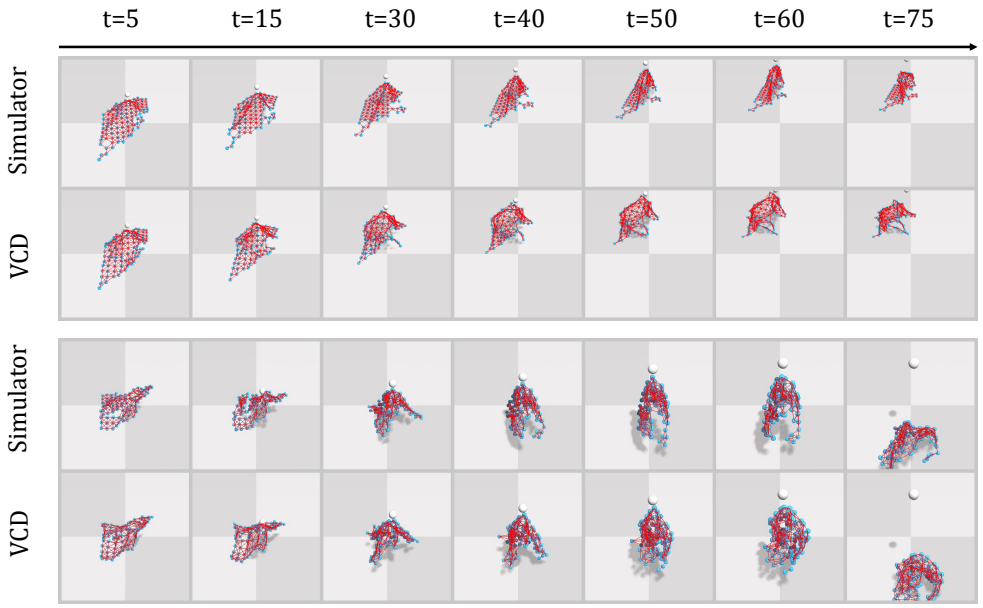
Supplementary Table 4: Normalized coverage (NC) of VCD in the real world.

D.2.3 Visualization of Sampled Actions in The Real World

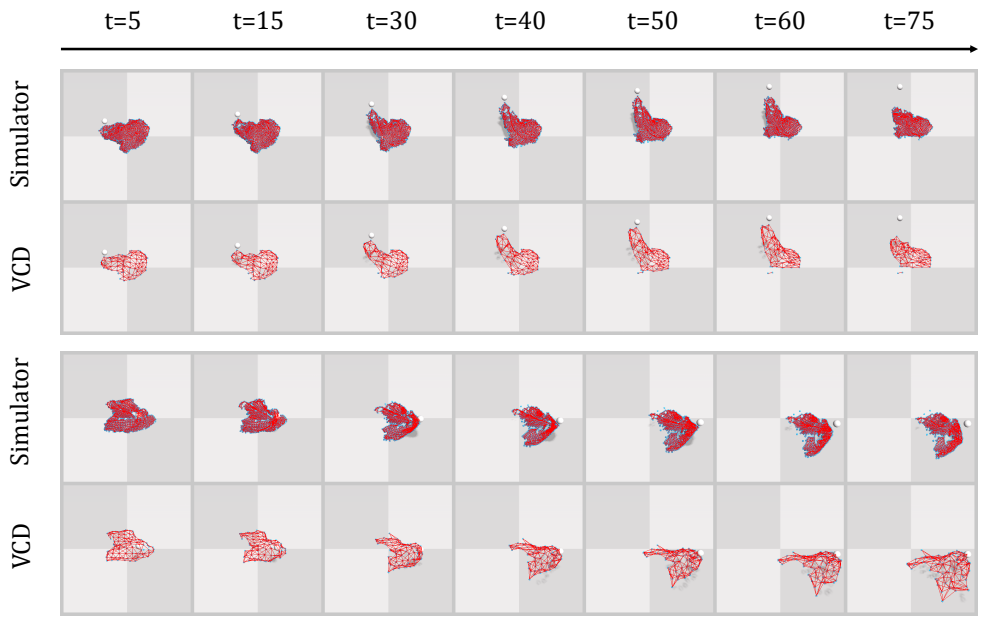
We show in Supplementary Figure 9 VCD's predicted score for each of the sampled action during smoothing of the cloth. Interestingly, though there is no explicit optimization for this, VCD favours picking corner or edge points and pulling outwards, which is a very natural and effective strategy for smoothing. This demonstrates the effectiveness of VCD for planning.



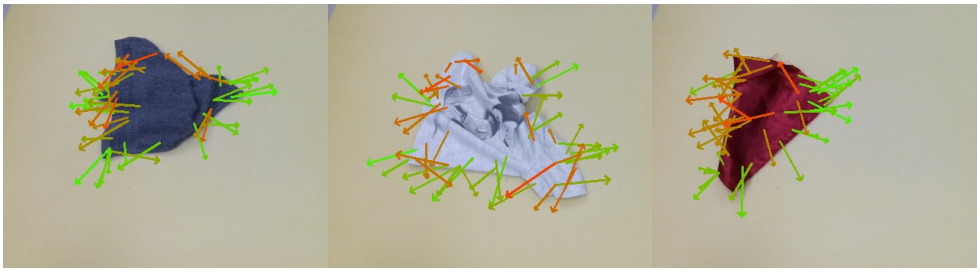
Supplementary Figure 6: Two open-loop predictions of VCD on square cloth. Blue points are observable particles/point cloud points and red lines are mesh edges. For each prediction, the top row is the ground-truth observable particles connected by the ground-truth mesh edges in simulator. The bottom row is the predicted point clouds by VCD, in which the mesh edges are inferred by the edge prediction GNN.



Supplementary Figure 7: Two open-loop predictions of VCD on rectangular cloth. Note VCD is only trained on square cloth. Blue points are particles/point cloud points and red lines are mesh edges. For each prediction, the top row is the ground-truth observable particles connected by the ground-truth mesh edges in simulator. The bottom row is the predicted point clouds by VCD, in which the mesh edges are inferred by the edge prediction GNN.



Supplementary Figure 8: Two open-loop predictions of VCD on t-shirt. Blue points are particles/point cloud points and red lines are mesh edges. Note VCD is only trained on square cloth. For each prediction, the top row is the ground-truth observable particles connected by the ground-truth mesh edges in simulator. The bottom row is the predicted point clouds by VCD, in which the mesh edges are inferred by the edge prediction GNN.



Supplementary Figure 9: Examples of 50 sampled actions used for planning. Each arrow goes from the 2D projection of the pick location to that of the place location. The actions with the higher predicted reward are shown in greener color and the actions with the lower predicted reward are shown in redder colors.

E Planning with VCD for Cloth Folding

We show that VCD can also be used for cloth folding. We assume an initially flattened cloth is given, which can be obtained via planning with VCD for smoothing. Given a goal configuration of a target folded cloth (e.g., a diagonal fold for square cloth), we use VCD with CEM to plan actions that fold the cloth into the target configuration. We explore the following three different goal specifications and cost functions for the CEM planning:

- A ground-truth cost function and goal specification that assumes access to the simulator cloth particles. The goal configuration of the cloth is specified as the goal locations of all particles. Given the voxelized point cloud of the initially flattened cloth, we first find a nearest neighbor mapping from each point in the point cloud to the simulator particles. The cost is then computed as the distance between the points in the achieved point cloud and their corresponding nearest-neighbor particles in the goal configuration.
- We use the point cloud of the cloth for goal specification and Chamfer distance as the cost. Specifically, the cost is the Chamfer distance between the achieved point cloud and the goal point cloud.
- We use the depth image of the cloth for goal specification and 2D IOU as the cost. Specifically, we compute the intersection over union between the segmented achieved depth map and the segmented goal depth map as the cost.

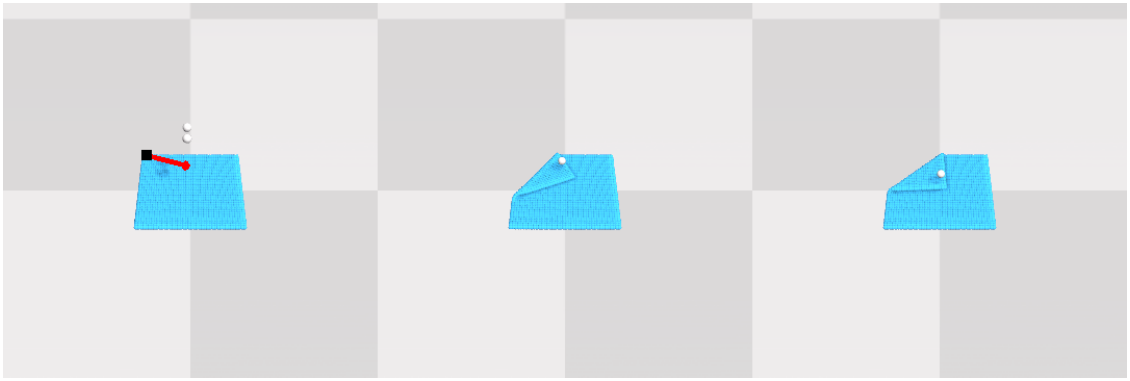
We evaluate VCD on three goals as shown in Supplementary Figure 10, 11, 12: (1) one-corner-in, which folds one corner of the square cloth towards the center; (2) diagonal, which folds one corner of the square cloth towards the diagonal corner; (3) arbitrary, which folds one corner of the square cloth towards the middle point of the opposite edge. For evaluation, we report the average particle distance between the achieved cloth state and the goal cloth state. The numerical results are shown in Supplementary Table 5 and the qualitative results are shown in Supplementary Figure 10, 11, 12.

As the result shows, VCD can be applied for folding with the above three ways for goal specification. For the ground-truth goal specification and cost computation using simulator particles, VCD performs fairly well for folding (average particle error within 0.3 - 1.3 cm, also see Supplementary Figure 10, 11, 12 for qualitative results). With goal specification via point cloud and Chamfer distance as the cost, the performance of VCD is also reasonable (average particle error 0.3 - 2 cm, also see below for qualitative results), making it a practical choice to apply VCD for folding in the real world.

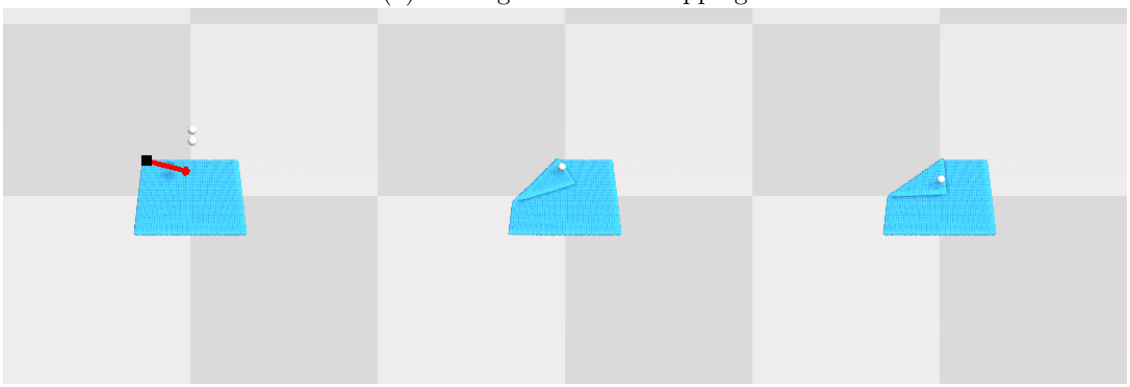
We also note that this VCD model is trained with random pick-and-place actions; the folding performance could be further improved if we add bias (such as corner grasping) during data collection to train VCD with more folding motions.

	One-corner-in	Diagonal	Arbitrary
Ground-truth mapping	3.480	13.466	4.136
Chamfer distance	3.311	19.398	18.132
IOU	36.897	15.744	19.871

Supplementary Table 5: Average particle distance (mm) between final achieved cloth state and goal cloth state.



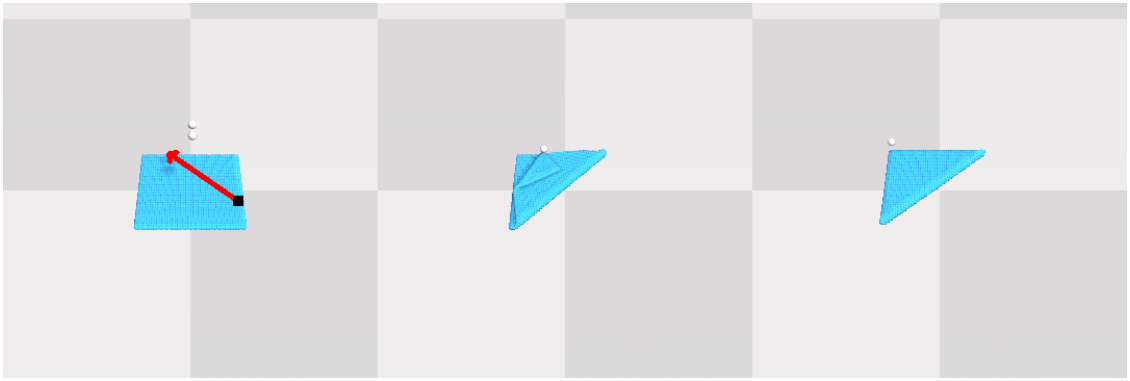
(a) Cost: groundtruth mapping



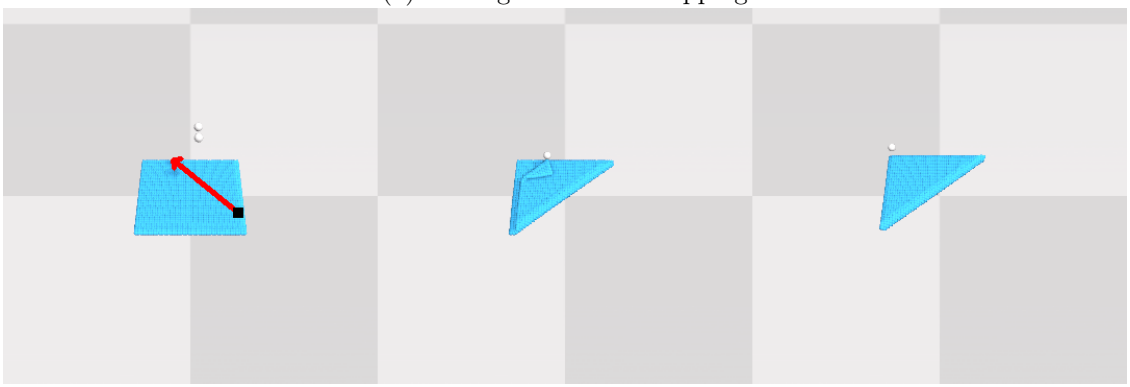
(b) Cost: Chamfer distance

(c) Cost: IOU

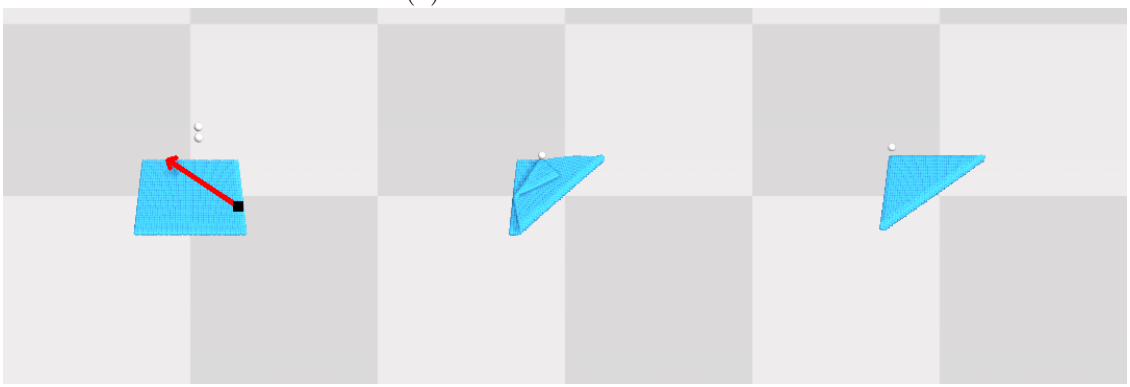
Supplementary Figure 10: VCD for folding, one-corner-in goal. The left column is the planned action, the middle column is the final achieved cloth state, and the right column is the goal.



(a) Cost: groundtruth mapping

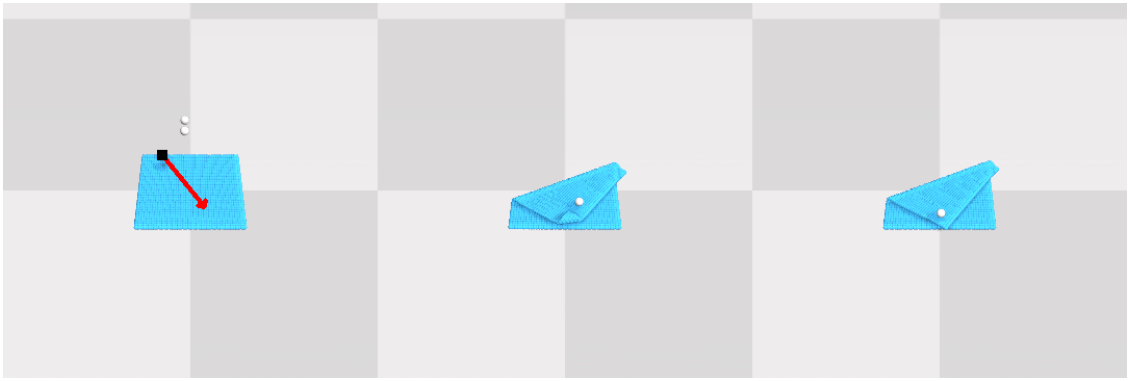


(b) Cost: Chamfer distance

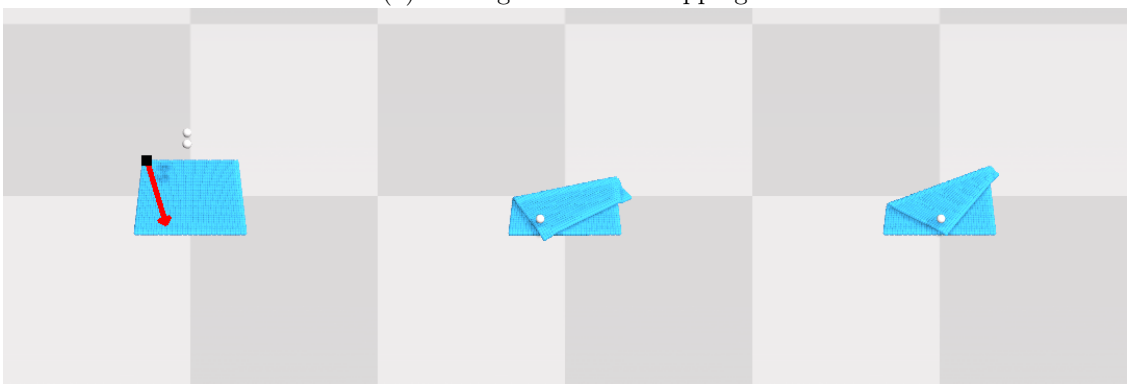


(c) Cost: IOU

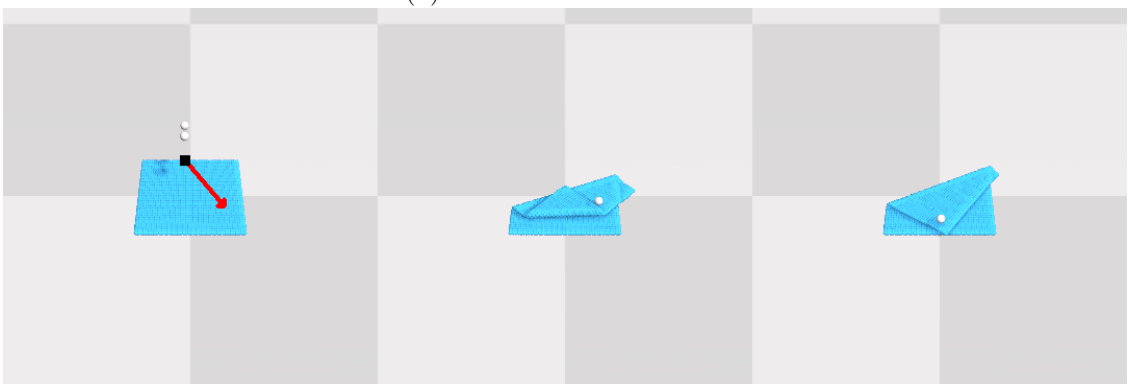
Supplementary Figure 11: VCD for folding, diagonal goal. The left column is the planned action, the middle column is the final achieved cloth state, and the right column is the goal.



(a) Cost: groundtruth mapping



(b) Cost: Chamfer distance

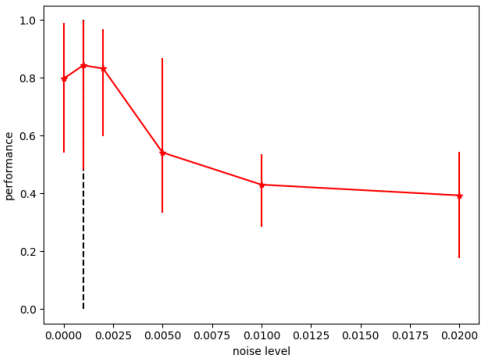


(c) Cost: IOU

Supplementary Figure 12: VCD for folding, arbitrary goal. The left column is the planned action, the middle column is the final achieved cloth state, and the right column is the goal.

F Robustness to Depth Sensor Noise

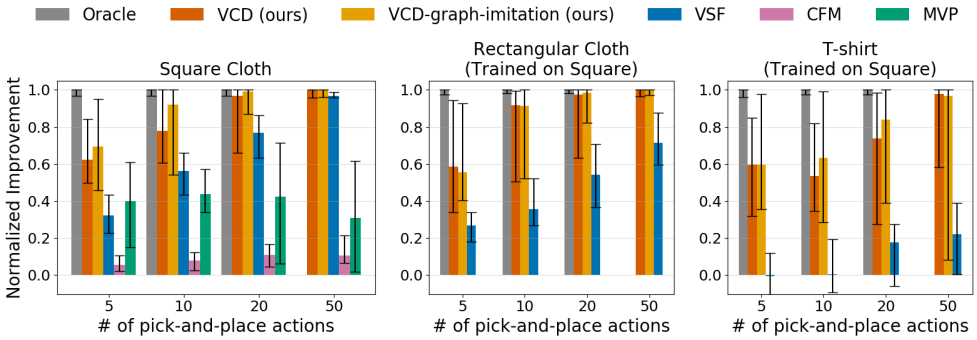
When deployed in the real world, VCD might suffer from the depth camera noise. To investigate this, we manually add different levels of noise (Gaussian noise with different levels of variance) to the depth map in the simulation and test VCD's planning performance (with a maximal number of 10 pick-and-place actions). The result is shown in Supplementary Figure 13. The dashed vertical line is the noise level of Azure Kinect depth camera that we use in the real world, as measured by Michal et al. [10]. As shown, VCD is quite robust within the noise range of the Azure Kinect depth sensor.



Supplementary Figure 13: Normalized Improvement of VCD under different levels of depth sensor noise, with a maximal number of 10 pick-and-place actions for smoothing. The vertical dashed line represents the typical level of Azure Kinect noise, which is the depth sensor that we use for the real-world experiment. The error bars show the 25% and 75% percentile.

G Comparison to Oracle using the FleX Cloth Model

How good can the system be if we know the full cloth dynamics? To answer this question, for our simulation experiments (shown in Supplementary Figure 14), we additionally show the performance of an oracle that uses the FleX cloth model for planning in Supplementary Figure 14. Here, oracle uses the same planning method as VCD and achieves perfect results in different clothes. This shows that better performance can be achieved if the full cloth model and dynamics can be better estimated, which we leave for future work.

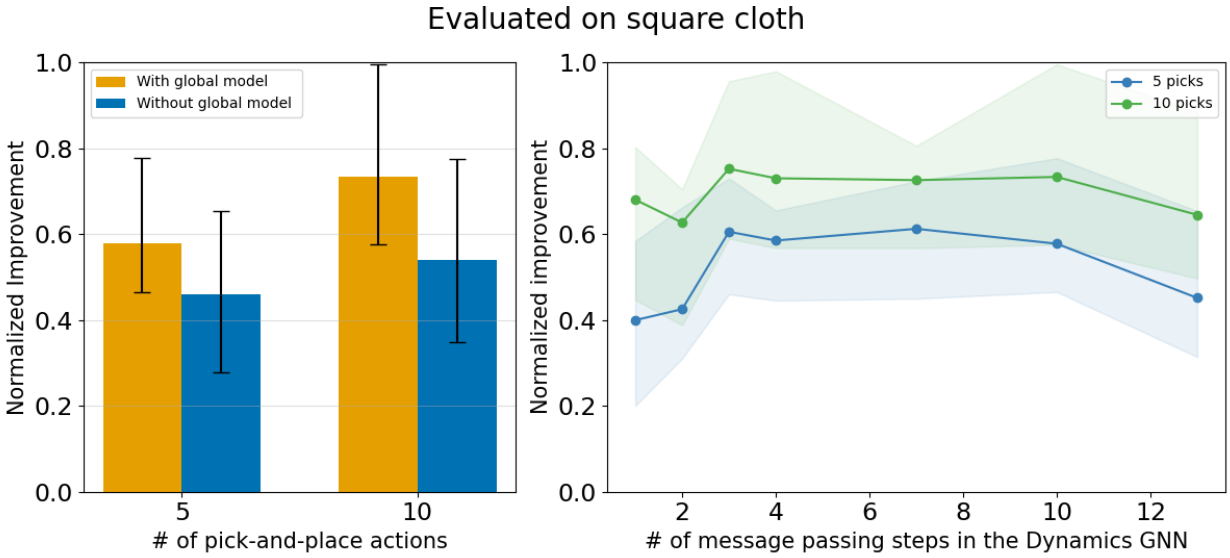


Supplementary Figure 14: Normalized improvement on square cloth (left), rectangular cloth (middle), and t-shirt (right) for varying number of pick-and-place actions. The height of the bars show the median while the error bars show the 25 and 75 percentile.

H Ablations on architectural choices

For our edge and dynamics GNNs, we adopt the model architecture from GNS [1], as described in Appendix A.1. In Sanchez-Gonzalez, et al [1], a comprehensive analysis on architectural design decisions for the GNS model was investigated. We modify the GNS architecture by adding a global model in each GN block of the processor, which has the potential to speed up the propagation of information across the graph. The global model has been widely used in previous works in graph neural networks [2, 11, 12]. Supplementary Figure 15 (left) shows that using a global model in the dynamics model yield better planning performance than without it.

We also evaluate the sensitivity of our dynamics model to the number of message passing steps (L). As shown in the right figure of Supplementary Figure 15, our dynamics model is robust to a broad range of values for the number of message passing steps. We speculate that, when the number of message passing is too small, the effect of action cannot propagate to the particles that are distant from the picked point. With too many message passing steps, the model is prone to overfitting. Nonetheless, Supplementary Figure 15 (right) shows that there is a broad of values for the number of message passing steps that lead to similar performance; thus, our model is fairly robust to this parameter.



Supplementary Figure 15: We evaluate the effects of a global model and the number of message passing steps in the dynamics GNN on the square cloth. The left figure shows that the usage of a global model is helpful to the planning performance. The right figure shows that our model is generally robust to the number of message passing steps as long as the number lies within the range of [3, 10].

References

[1] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Sci Robot*, 5(47), October 2020.

[5] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[6] Xingyu Lin, Yufei Wang, Jake Olkin, and David Held. SoftGym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, 2020.

[7] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation. In *Robotics: Science and Systems (RSS)*, 2020.

[8] Wilson Yan, Ashwin Vangipuram, Pieter Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation. In *Conference on Robot Learning (CoRL)*, 2020.

[9] Wilson Wu, Yilin adn Yan, Thanard Kurutach, Lerrel Pinto, and Pieter Abbeel. Learning to manipulate deformable objects without demonstrations. *Robotics Science and Systems (RSS)*, 2020.

[10] Michal Tölgyessy, Martin Dekan, L'uboš Chovanec, and Peter Hubinskỳ. Evaluation of the azure kinect and its comparison to kinect v1 and kinect v2. *Sensors*, 21(2):413, 2021.

[11] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. Global context enhanced graph neural networks for session-based recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 169–178, 2020.

[12] Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.