1 Appendix

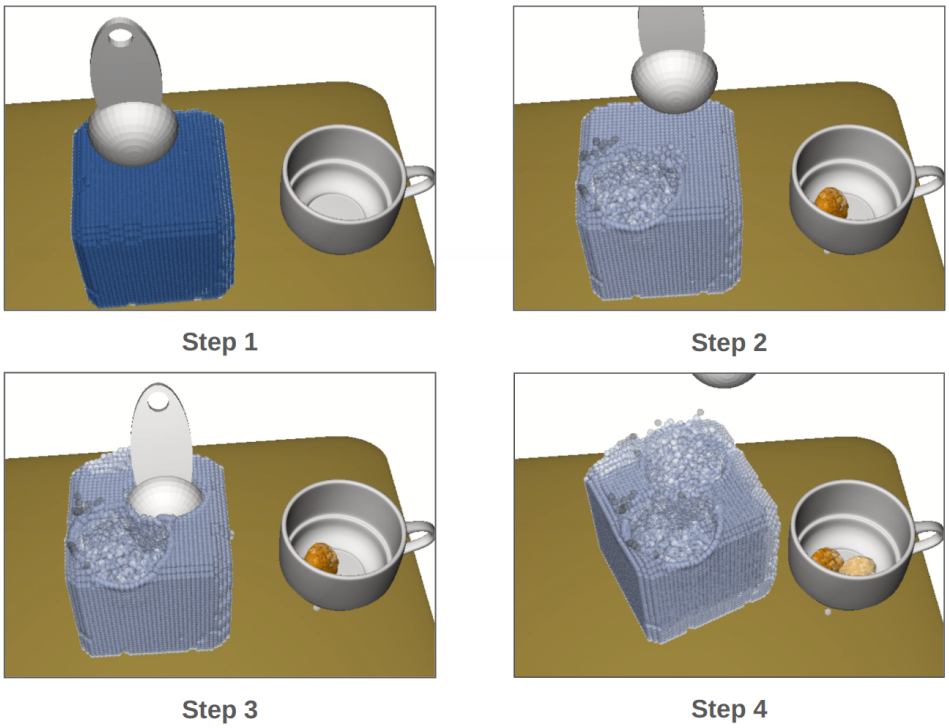

Figure 1: Ice–Cream Scooping Task. (Step 1) Initial setup of the scene, with the spoon positioned above the ice cream cube. A white cup is placed beside the ice cream block to receive the scooped pieces. (Step 2) After the first scoop, a distinct ice cream piece appears in the cup, shown in a different color to indicate it is a separate connectivity cluster. (Step 3) The spoon performs a second scooping, inserting into the ice cream cube to gather another piece. (Step 4) The task is completed with two distinct ice cream pieces, each in its own cluster, now present in the cup.

1.1 Additional Manipulation Tasks

While our main focus is on multi-step cutting, our simulator, topology discovery mechanism, and spectral reward function extend naturally to other deformable-object tasks. Below, we describe two illustrative examples that highlight the potential for extending our framework, using enumerated steps for each task's execution.

1.1.1 Cream Writing

In this task, the objective is to "write" a target word (e.g., `CORL`) by extruding a soft, von Mises-plastic "cream" material onto a flat surface. The cream is modeled in MPM with low yield stress and high plasticity, allowing it to retain its shape after extrusion. The agent controls a "cream pen" at a fixed height above the table. Its state is represented by its (x, y) position, and the action is a 3-dimensional vector $(\Delta x, \Delta y, b)$, where $(\Delta x, \Delta y)$ specifies the pen movement and $b \in \{0, 1\}$ toggles the extrusion.

During the writing process, each continuous stroke formed while $b = 1$ is considered a single connectivity cluster. If the pen is lifted and re-positioned, the next stroke is assigned to a new cluster. Thus, the clustering mechanism is naturally defined by the writing process, not by explicit color control. The visualization in Figure 2 uses different colors to indicate separate clusters, a distinction that becomes critical when computing the spectral-based reward.

Execution steps:

1. Position the pen at the starting point for the letter "C," toggle $b = 1$, and move rightward to trace the stroke, forming the first connectivity cluster.

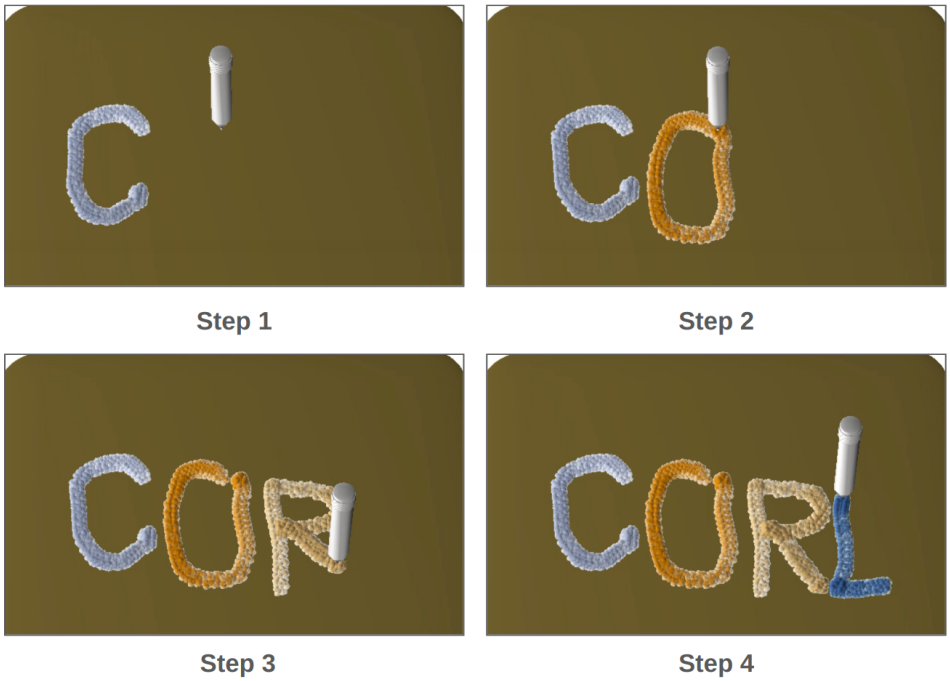

Figure 2: Cream Writing Task. (Step 1) The cream pen extrudes a continuous stroke to form the letter "C." (Step 2) The pen lifts, repositions, and extrudes the letter "O." (Step 3) Next, it draws "R." (Step 4) Finally, it writes "L." Each letter is rendered in a different color to indicate that they belong to separate connectivity clusters. The color differentiation is not manually controlled but automatically generated based on the clustering, which is leveraged during the spectral-based reward computation.

2. Lift the pen, move to the position for the letter "O," toggle $b = 1$, and extrude to form the second cluster.
3. Reposition to the third slot, toggle $b = 1$, and draw the letter "R," creating a new cluster.
4. Finally, move to the fourth slot, toggle $b = 1$, and write the letter "L," forming the final cluster.

Our spectral reward function evaluates how well each cluster corresponds to the intended letter shape. This clustering mechanism provides a natural way to segment distinct components, allowing the reward function to assess both spatial structure and connectivity. However, the sequential planning of strokes remains a complex decision-making challenge, distinct from the cutting task, and thus extending PDDP to handle such "additive" tasks is a promising direction for future work.

1.1.2 Ice-Cream Scooping

In this task, the objective is to scoop a chunk from a block of "pudding"-like ice cream (elastoplastic MPM) and deposit it into a cup. The agent controls a spoon at a fixed height, using a 2D (x, y) action space to position the scoop. Once positioned, the agent executes a predefined scooping primitive as follows:

Execution steps:

1. Lower the spoon into the ice cream block to initiate contact.
2. Translate the spoon forward to penetrate and gather the material.
3. Lift the spoon while rotating it to maintain a horizontal orientation, preventing spillage.
4. Move the spoon above the cup.
5. Tilt or open the spoon to release the scooped chunk into the cup.

The scooped chunk forms a distinct connectivity cluster, separate from the remaining block. As illustrated in Figure 1, each scooped piece is automatically assigned a different color to indicate it belongs to a separate cluster. This distinction is crucial when computing our spectral-based reward, as it allows for quantifying successful segmentation and accurate placement of the scooped pieces.

1.1.3 Discussion

These two examples demonstrate the extensibility of our MPM-based environment, topology discovery, and spectral reward function to a broader range of deformable manipulation tasks beyond multi-step cutting. The use of distinct clusters to represent separate connectivity components is a key mechanism for computing spectral-based rewards, applicable in both "additive" and "scooping" tasks. Designing specialized action primitives and learning-based planners for these scenarios remains a promising direction for future exploration.

1.2 Pyramid Cutting Task

Motivation and Objective. – To further evaluate the generalization capability of our spectral reward function to more complex and arbitrary goal shapes, we introduce the pyramid cutting task (Figure 3). In contrast to the predefined slicing, sticking, and dicing tasks, this experiment requires the agent to carve out a corner segment from a cubic block to produce a pyramid-shaped fragment. The target shape is defined as a triangular pyramid with sloped surfaces, representing a more intricate and asymmetrical geometry than previous tasks.

The objective of this experiment is to assess whether the MPPI planning method, driven solely by our spectral reward function, can effectively discover a plausible cutting trajectory to achieve the desired pyramid shape without any task-specific tuning or retraining. This setup challenges the reward function to guide cutting actions that align with more complex, multi-faceted surfaces, testing its robustness and generalizability.

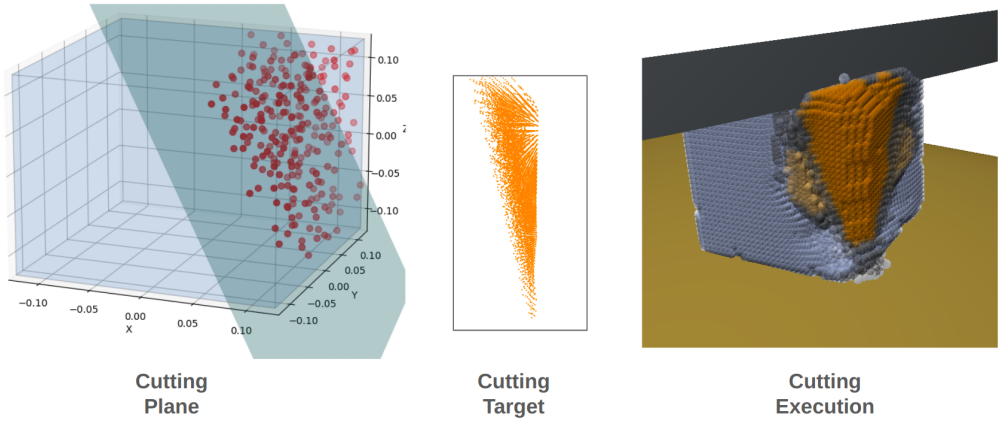

Figure 3: Pyramid Cutting Task. (Left) Cutting Plane: Visualization of the cutting plane planned by the MPPI controller based on our spectral reward function. The plane is strategically positioned to carve out a pyramid-shaped fragment from the corner of the cube. (Middle) Cutting Target: The target goal shape, shown as a point cloud representation, highlighting the desired pyramid structure that serves as a guide for the cutting task. (Right) Cutting Execution: The scene depicts the execution of the planned cutting trajectory, where the knife removes the corner section to achieve the specified pyramid shape. The orange particles indicate the material identified as the target segment to be removed, aligning with the planned cutting plane.

Takeaway. – The results of the pyramid cutting task clearly demonstrate that our spectral reward function effectively generalizes to more intricate geometric goals, even in cases involving angled and asymmetric surfaces. As shown in Figure 3, the MPPI planner successfully identifies a cutting plane that produces a distinct pyramid-shaped fragment from the cube's corner, aligning well with the target structure. The qualitative alignment between the planned cutting plane and the final extracted segment validates that the reward function remains a reliable guidance signal, even for complex and non-standard shapes. This outcome highlights the adaptability of our framework, suggesting its potential to handle more sophisticated and arbitrary cutting tasks in future applications.

1.3 Perception Model Architecture Details

In practice, we design two distinct styles of perception models to explore their effectiveness in our setup: a graph-based model and a point-based model. The graph-based model leverages graph convolutional networks (GCNs) and joint attention mechanisms to process structured graph data, while the point-based model employs a hierarchical PointNet-style architecture using SPALPA blocks to handle point cloud data. After extensive evaluation, we adopt the graph-based model as our final choice. This decision is driven by the significant reduction in parameter count (approximately four times fewer parameters) and faster training and inference speed compared to the point-based model. The PointNet-based architecture, though capable of capturing finer local features, incurs substantial computational overhead due to its deeper SPALPA structure and multi-scale processing layers.

1.3.1 PointNet-Based Perception Model Architecture

The PointNet-based architecture processes point cloud data through hierarchical SPALPA blocks for multi-scale feature extraction. The key components are:

- **Topological Encoder:** The primary perception encoder in this architecture, responsible for extracting features from 32-dimensional topological inputs. It comprises 5 SPALPA blocks that progressively increase feature dimensions as follows:

$$96 \rightarrow 192 \rightarrow 384 \rightarrow 768 \rightarrow 1536$$

  Each SPALPA block includes:
  - **Local Attention:** Extracts localized spatial relationships through ‘Conv2d‘.
  - **Global Attention:** Aggregates broader context using multi-scale convolutional layers.
  - **Grouper:** Implements ‘QueryAndGroup‘ for spatial neighborhood aggregation.
- **Action Encoder:** Encodes 2-dimensional action inputs using the same hierarchical structure as the Topological Encoder, ensuring feature alignment and consistency.
- **Cross-Attention Network:** Integrates topological and action features through CrossAttnNet modules, employing PreNorm Attention, cross-attention, and gated residual connections.
- **Decoder:** Reconstructs the feature maps and propagates them to the original point cloud resolution using ‘FeaturePropagation‘ blocks, merging multi-scale features.
- **Output Heads:**
  - **Segmentation Head:** Projects features to segmentation classes using ‘Conv1d‘ layers.
  - **Point Cloud Head:** Predicts point coordinates through residual blocks and a final ‘Linear‘ layer.

  Despite its comprehensive feature extraction capability, the PointNet-based model incurs considerable computational overhead, resulting in slower training and inference compared to the graph-based model.

1.3.2 Graph Network Architecture

The Graph Network architecture processes structured graph data using GCN layers and attention-based mechanisms. The core perception module in this architecture is the **t_graph_encoder**, which encodes topological information through GCN layers.

- **t_graph_encoder:** The designated perception module in this architecture, processing node and label embeddings through GCN layers structured as:

$$3 \rightarrow 96 \rightarrow 96$$

  Key submodules include:
  - **Node Encoder:** Projects 3-dimensional node features to a 96-dimensional latent space.
  - **Label Encoder:** Processes 32-dimensional node labels through GCN layers.
  - **Embedding Encoder:** Integrates 192-dimensional precomputed embeddings into a unified 96-dimensional space.
- **a_graph_encoder:** Processes action-related graph data using a similar structure to the ‘t_graph_encoder‘, but with 2-dimensional action inputs.
- **Joint Graph Transformer:** Integrates features from both graph encoders through multi-layer attention and cross-attention modules:

$$96 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4$$

Table 1: Empirical Studies of Pretraining Strategies for Perception Models: We evaluate the effectiveness of graph-based and point-based perception models under various pretraining strategies, including sorting the input point cloud, applying Gaussian noise for data augmentation, and downsampling the input points before processing. The final model selected for deployment is the Graph-Based model with sorting, Gaussian noise, and 256 downsampled points, balancing accuracy and computational efficiency.

Perception Model	With Sort	Gaussian Noise	Downsample Points	Seen Accuracy	Unseen Accuracy
Graph-Based	✗	✗	128	66.4	48.7
Graph-Based	✓	✗	128	73.8	52.1
Graph-Based	✓	✓	128	75.4	69.3
Graph-Based (Final)	✓	✓	**256**	**77.1**	**74.1**
PointNet-Based	✗	✓	2048	73.2	61.2
PointNet-Based	✓	✓	512	79.2	75.4

- **Query Point Encoder:** Processes query nodes in the graph structure through ‘Linear‘ and ‘ReLU‘ layers:

$$3 \rightarrow 64 \rightarrow 96$$

- **Output Heads:**
  - **Node Position Head:** Outputs node coordinates through residual blocks.
  - **Node Feature Head:** Projects node features to a 32-dimensional space.

  The graph-based model provides a more parameter-efficient and computationally feasible architecture while maintaining robust feature extraction and representation capabilities, making it the preferred choice in our implementation.

Summary: – We explore both PointNet-based and Graph Network architectures to identify the optimal perception model for our setup. While the PointNet-based model employs extensive SPALPA blocks for multi-scale feature extraction, its computational cost and parameter count are substantially higher than the graph-based model. Consequently, we adopt the graph-based architecture, leveraging the **t_graph_encoder** as the primary perception module due to its significantly lower parameter count, faster training and inference speeds, and effective integration of topological and action features.

1.3.3 Empirical Evaluation of Perception Models

Discussion and Model Selection: – Table 1 presents a comparative evaluation of graph-based and point-based perception models under various pretraining strategies. The primary factors analyzed include sorting based on the x-axis, Gaussian noise for data augmentation, and the degree of point cloud downsampling.

- **Sorting and Gaussian Noise:** Incorporating sorting and Gaussian noise consistently improves model accuracy for both seen and unseen data. For the graph-based model, enabling both sorting and Gaussian noise increased unseen accuracy from 48.7% to 69.3% under a 128-point downsampling setting.
- **Downsampling Strategy:** The impact of point cloud resolution is evident as increasing the number of points from 128 to 256 in the graph-based model further improved unseen accuracy to 74.1%. This result underscores the importance of maintaining sufficient point density to preserve critical spatial information.
- **Comparison of Models:** The point-based model with 512 points achieves 75.4% unseen accuracy, slightly outperforming the graph-based model with 256 points. However, the computational overhead associated with the point-based model is significantly higher, consuming approximately four times more parameters and resulting in slower training and inference.

Final Model Selection: – Based on the empirical findings, the graph-based model with sorting, Gaussian noise, and 256 downsampled points is selected as the final perception model for deployment. This configuration achieves a balanced trade-off between computational efficiency and segmentation accuracy, making it a practical choice for large-scale robotic manipulation tasks.

1.4 Policy Model Architecture Details

In our framework, the policy model architecture is structured to handle skill-conditioned action generation through a structured diffusion process. The model, named **PDDP (Particle-Based Diffusion Policy)**, leverages adaptive normalization, timestep embeddings, and modular diffusion blocks to effectively model action trajectories. Notably, the observation encoder in PDDP utilizes the perception encoder previously defined in the **Perception Model Architecture** section, ensuring consistent feature extraction and representation across the pipeline.

1.4.1 Policy Model Architecture - PDDP

The PDDP architecture is structured as follows:

1. Encoders: –

- **Observation Encoder:** The observation encoder adopts the perception encoder design outlined previously, specifically utilizing the graph-based **t_graph_encoder** as the primary perception module. This encoder processes topological node features through a GCN and includes three key submodules:
  - **Node Encoder:** Projects 3-dimensional node features to a 96-dimensional latent space through a 2-layer GCN:
    $$3 \to 96 \to 96$$
  - **Label Encoder:** Transforms 32-dimensional node labels to a 96-dimensional space using the same GCN structure.
  - **Embedding Encoder:** Integrates 192-dimensional precomputed embeddings, projecting them to 96 dimensions.

  The outputs from each encoder component are processed through a shared 'LayerNorm' to stabilize feature representation.
- **Goal Encoder:** Encodes 3-dimensional goal vectors through a fully connected network:
  $$3 \to 64 \to 96$$
  ReLU activation is applied to the intermediate layer.
- **Timestep Embedding (sigma_map):** Encodes the diffusion timestep using a 2-layer MLP:
  $$256 \to 128 \to 128$$
  A SiLU activation is applied after the first linear layer to maintain smooth gradient flow.
- **Action History Layer:** Integrates action history features through a fully connected layer:
  $$21 \to 128$$
- **Conditional Layer:** Aggregates timestep, goal, and action history embeddings into a unified feature space:
  $$256 \to 128$$

2. Backbone - Diffusion Blocks (DDiTBlock): –

The core processing backbone of PDDP consists of a sequence of 12 **DDiTBlocks**. Each DDiTBlock contains:

- **Self-Attention Layer:** Applies self-attention to the feature space with 96-dimensional query, key, and value projections:
  $$96 \to 288 \to 96$$
  This allows the model to capture dependencies across all points in the point cloud, facilitating information exchange across nodes.
- **MLP Layer:** Implements a 2-layer MLP with GELU activation:
  $$96 \to 384 \to 96$$
  This structure refines the feature representation after the attention operation, maintaining non-linear feature transformations.

- **Dropout:** Applied to both the attention and MLP layers with a probability of 0.4 to mitigate overfitting and stabilize training.
- **Adaptive LayerNorm (adaLN):** Adaptive LayerNorm is applied to each block using modulation inputs derived from both the timestep embedding and goal encoder. The modulation layer is structured as:

$$128 \rightarrow 576$$

  This mechanism enables each DDiTBlock to dynamically adjust its feature scaling based on task-specific conditions.

3. Output Layer: –

The final output layer processes the feature representation generated by the diffusion blocks to predict binary classification logits for each point in the point cloud. The output shape is defined as:

$$\text{Output Shape: } [\text{batch size}, \text{num points}, 2]$$

A linear projection is applied to transform the 96-dimensional feature space into 2-dimensional logits, representing the binary classification output for each point. This design aligns the output structure with the expected action space while maintaining consistency across the diffusion layers.

Summary: – The PDDP architecture is structured to leverage a modular pipeline consisting of encoders for observation, goal, and timestep embeddings, a backbone of 12 DDiTBlocks with adaptive normalization and self-attention mechanisms, and a binary classification output layer for per-point action prediction. The use of the **t_graph_encoder** as the primary perception module ensures consistent feature extraction and integration across both the perception and policy networks, promoting robust action generation in diverse manipulation tasks.

1.5 Material Point Method in Details

The Material Point Method (MPM) is a hybrid Lagrangian–Eulerian scheme originally introduced by Sulsky *et al.* for solid mechanics. In MPM, state variables (mass, momentum, deformation) are carried on material points (particles), while a fixed Eulerian grid is used to solve the equations of motion. This split representation combines the mesh-free advantages of particle methods with the stability and boundary-handling capabilities of grid methods.

Governing Equations – MPM begins from the continuum balance laws in Eulerian form:

$$\frac{D\rho}{Dt} + \rho\,\nabla\cdot v = 0, \qquad \rho\,\frac{Dv}{Dt} = \nabla\cdot\sigma + \rho\,g,$$

where ρ is the mass density, v the velocity field, σ the Cauchy stress tensor, and g the body-force (e.g. gravity). The material derivative D/Dt captures convection of field quantities with the flow.

Weak Form and Discretization – To derive a tractable discretization, one multiplies the momentum equation by a test function $q(x)$ and integrates over the domain Ω^n at time step n. Integration by parts moves spatial derivatives onto q, yielding the weak form. We then partition Ω^n into particle subdomains Ω_p^n, and approximate both trial and test functions using B-spline shape functions $N_i(x)$ centered at grid nodes i. This Galerkin-style projection leads to discrete nodal equations that are assembled via sums over particles.

Particle–to–Grid (P2G) Transfer – Each particle p holds:

$$\{\, m_p,\ V_p,\ v_p,\ F_p\}$$

for mass m_p, volume V_p, velocity v_p, and deformation gradient F_p. To project onto the grid, we compute at each node i:

$$m_i \;=\; \sum_p m_p\, N_i(x_p), \quad (mv)_i \;=\; \sum_p m_p\, v_p\, N_i(x_p), \quad f_i^{\text{int}} \;=\; -\sum_p V_p\, \sigma_p\, \nabla N_i(x_p),$$

where σ_p is obtained from the chosen constitutive law (e.g. hyperelastic) evaluated at F_p. These transfers ensure exact conservation of mass and momentum.

Grid Update – On the Eulerian grid, we update nodal momentum via a symplectic (explicit) Euler step:

$$(mv)_i^{n+1} = (mv)_i^n + \Delta t(f_i^{\text{int}} + m_i\, g), \qquad v_i^{n+1} = \frac{(mv)_i^{n+1}}{m_i}.$$

This step advances velocities under both internal stresses and external forces, while preserving stability for moderate time steps.

Grid–to–Particle (G2P) Transfer – After updating the grid, we interpolate back to particles:

$$v_p^{n+1} = \sum_i N_i(x_p)\, v_i^{n+1}, \qquad C_p = \sum_i v_i^{n+1}\, \nabla N_i(x_p)^T,$$

where C_p is an affine velocity gradient matrix (used in variants like APIC) that captures sub-cell velocity variation.

Deformation Gradient Update – The particle deformation gradient evolves according to

$$\frac{dF}{dt} = (\nabla v)\, F.$$

Using the interpolated velocity gradient C_p, we discretize:

$$F_p^{n+1} = (I + \Delta t\, C_p)\, F_p^n,$$

so that F_p accumulates the local deformation history on each particle.

Stress Computation – For hyperelastic materials, one defines a strain-energy density $\Psi(F)$ and computes the first Piola–Kirchhoff stress

$$P_p = \frac{\partial \Psi}{\partial F}(F_p),$$

then the corresponding Cauchy stress

$$\sigma_p = \frac{1}{\det F_p}\, P_p\, F_p^T.$$

This stress is used in the P2G transfer to produce internal forces.

Algorithm Summary – At each timestep, MPM executes:

1. **P2G:** Transfer $\{m_p,\ v_p,\ F_p\}$ to grid nodes $\{m_i,\ v_i,\ f_i\}$.
2. **Grid Update:** Integrate nodal momentum, compute v_i^{n+1}.
3. **G2P:** Interpolate v_i^{n+1} and ∇v back to particles.
4. **State Update:** Update each F_p^{n+1} and compute σ_p.
5. **Advection:** Move particles: $x_p^{n+1} = x_p^n + \Delta t\, v_p^{n+1}$.
6. **Reset:** Clear grid variables for the next iteration.

Thanks to its hybrid nature, MPM can robustly simulate extreme deformations, fracture propagation, and multi-body contact without remeshing, making it a powerful tool in graphics, engineering, and robotic manipulation contexts.

1.6 Particle-based Damage-tracking and Topological Reconstruction in Details

We propose a robust particle-based method to accurately determine whether a cutting action successfully separates an object into discrete pieces. Our approach integrates particle-level damage tracking with topological surface reconstruction using the Material Point Method (MPM) as shown in Figure 4. Within our MPM framework, each particle's deformation state is computed, and we define a particle as "damaged" based on critical compression and stretch thresholds in its deformation gradient $\mathbf{F}$. Formally, a particle p is classified as damaged if:

$$J_p \leq (1 - \epsilon_c)^m \quad \text{or} \quad J_p \geq (1 + \epsilon_s)^m, \tag{1}$$

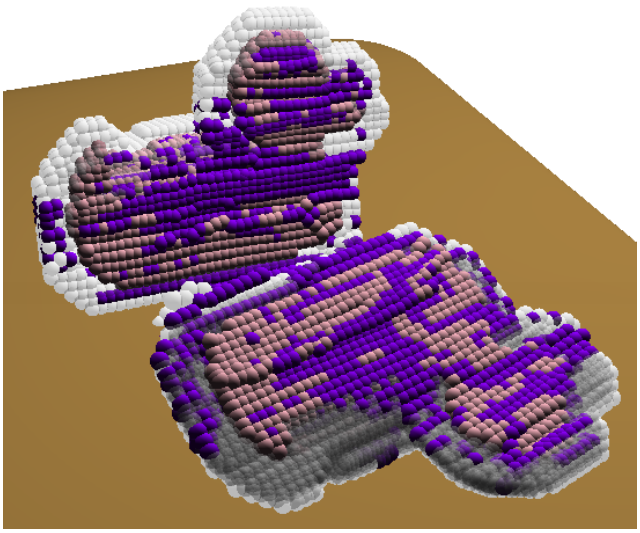

Figure 4: Particle-based damage-tracking and topological reconstruction during a cutting action. Gray and pink particles remain intact, while damaged particles (purple) mark the cut interface. From these damage signals we build an implicit SDF, extract the zero-isosurface via Marching Cubes, and then cluster connected components to recover discrete object segments.

where $J_p = \det(\mathbf{F}_p)$, ϵ_c, ϵ_s denote critical compression and stretch values, and m is a material-specific sensitivity exponent.

In practice, we select the compression and stretch thresholds ϵ_c and ϵ_s so that they correspond directly to the maximum volumetric change a particle can undergo before being flagged as damaged. Recall from Eq. (1) that

$$J_p = \det(\mathbf{F}_p)$$

measures the local volume ratio of particle p. For example, setting

$$\epsilon_c = 0.025, \qquad \epsilon_s = 0.01$$

means that a particle whose volume has decreased by more than 2.5% ($J_p \leq (1 - 0.025)^m$) or increased by more than 1% ($J_p \geq (1 + 0.01)^m$) is classified as damaged. These values are chosen to be small enough to detect the onset of fracture in brittle materials, yet large enough to avoid false positives under normal elastic deformation.

The exponent m modulates how sharply damage accumulates once J_p deviates from unity: a larger m yields an abrupt transition from "healthy" to "damaged," whereas a smaller m produces a more gradual damage accumulation. We calibrate ϵ_c, ϵ_s, and m using simulated uniaxial compression and tension tests: a small block is deformed at a constant strain rate, we record its volumetric Jacobian history, and then choose the smallest thresholds that cleanly separate reversible (elastic) behavior from irreversible damage.

For more ductile or plastic materials, one might increase the stretch threshold (e.g. $\epsilon_s \approx 0.05$) and use a lower sensitivity exponent (e.g. $m = 1$ or 2) to model gradual yielding. Conversely, for near-brittle media a tighter compression bound (e.g. $\epsilon_c \approx 0.01$) and higher exponent (e.g. $m \geq 4$) better capture sudden fracture. By anchoring these parameters to physically measurable strain limits, our framework remains both interpretable and readily tunable across a wide range of material behaviors.

For more complex yielding materials, such as those modeled with von Mises plasticity, damage occurs when the yielding stress σ_y falls below zero after particle softening:

$$\sigma_y^{(t+1)} = \sigma_y^{(t)} - \gamma \|\Delta\epsilon_p\|, \quad \text{with} \quad \text{Damaged}_p = \begin{cases} 1 & \text{if } \sigma_y^{(t+1)} \leq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where γ is the particle softening coefficient and $\Delta\epsilon_p$ represents incremental plastic strain.

Once particles become damaged, their mechanical properties are altered irreversibly, enabling consistent tracking of the damaged state throughout subsequent simulation steps.

To reconstruct and evaluate object topology following a cut, we perform explicit topological segmentation leveraging the particle damage signals and knife trajectory data. We track the knife trajectory precisely during interactions, ensuring a clean spatial separation between object segments. After the completion of a cut action, defined by a stationary knife and changed particle damage states, we reconstruct the object's surface mesh from particle data through an implicit surface representation using Signed Distance Fields (SDFs) and the Marching Cubes algorithm.

Formally, given the set of particles $\mathcal{P}$ and knife trajectory $\mathcal{T}$, we first construct an implicit SDF representation $SDF(\mathbf{x})$ of the object's spatial domain:

$$SDF(\mathbf{x}) = \min_{p \in \mathcal{P}} \|\mathbf{x} - \mathbf{x}_p\| - r_p, \tag{3}$$

where $\mathbf{x}_p$ is the position of particle p, and r_p is its effective influence radius. Critically, we incorporate the knife trajectory $\mathcal{T}$ by explicitly marking trajectory regions within the SDF, enforcing a strict gap devoid of particles. This guarantees that no particle fills the knife's trajectory space, creating a clear geometric boundary between separated pieces.

We perform surface reconstruction by extracting the zero-isosurface of the computed SDF using a GPU-accelerated Marching Cubes implementation provided by Warp [**?**]. Specifically, we first define a discrete volumetric grid around particle positions, evaluate the SDF values, and then generate vertices and faces for the surface mesh:

$$\text{vertices, faces} = \text{MarchingCubes}(SDF(\mathbf{x}) = 0). \tag{4}$$

This mesh reconstruction process ensures explicit representation of separated object components. After extracting mesh segments, we apply Laplacian smoothing to refine surface quality:

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \alpha \sum_{j \in \mathcal{N}(i)} (\mathbf{v}_j - \mathbf{v}_i), \tag{5}$$

where $\mathbf{v}_i$ is a vertex in the mesh, $\mathcal{N}(i)$ are neighboring vertices, and α is a smoothing factor.

Subsequently, we cluster these reconstructed mesh segments into distinct topological pieces by performing connected component analysis on the extracted mesh faces. Each particle is then assigned a cluster identity by evaluating its spatial relationship to these mesh segments. Given particle positions $\mathbf{x}_p$ and cluster meshes, we perform the following SDF-based assignment:

$$\text{Cluster}_p = \begin{cases} i & \text{if } SDF_i(\mathbf{x}_p) < \tau, \\ -1 & \text{otherwise,} \end{cases} \tag{6}$$

where SDF_i is the SDF for cluster mesh i, and τ is a proximity threshold.

After assigning new cluster identities, we update the global object topology by comparing current particle clusters to previously stored cluster identifiers. New cluster IDs are assigned whenever new separate segments emerge, thus preserving consistent object segmentation across simulation timesteps.

This methodical combination of particle damage tracking, implicit surface reconstruction, and mesh-based clustering provides a robust and efficient solution for detecting and managing the topological changes that occur during robotic cutting tasks. Our approach ensures accurate evaluation of cutting outcomes essential for effective policy learning and performance assessment.

1.7 Pose-Invariant Shape Evaluation via Spectral Analysis in Details

After discovering object topology, we require a robust metric to evaluate how closely the segmented fragments match the intended goal shape. Traditional point-set distances, such as Chamfer or Earth Mover's Distance, are sensitive to pose variations and require explicit alignment. To overcome these

limitations, we introduce a spectral-based reward derived from the intrinsic geometry of shapes, inherently invariant to rigid transformations.

Given a point cloud $X = \{x_i\}_{i=1}^n$, we construct a k-nearest neighbor graph where the edge weight between points i and j is defined as

$$W_{ij} = \exp\left(-\frac{d_{ij}^2}{\sigma^2}\right), \tag{7}$$

with d_{ij} the geodesic distance between points and σ a scaling parameter. We then build the degree matrix D and the combinatorial Laplacian:

$$D_{ii} = \sum_j W_{ij}, \quad L = D - W. \tag{8}$$

Eigen-decomposition of L yields the smallest k eigenpairs:

$$L\Phi = \Lambda\Phi, \tag{9}$$

where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_k)$ contains eigenvalues and $\Phi = [\phi_1, \ldots, \phi_k]$ are the corresponding eigenvectors. These spectral descriptors capture intrinsic shape information invariant to isometries.

Given two shapes X and Y, we define their spectral distance as:

$$d_{\text{spec}}(X, Y) = \alpha\|\Lambda_X - \Lambda_Y\|_2^2 + \beta\|\Phi_X^\top\Phi_X - \Phi_Y^\top\Phi_Y\|_F^2, \tag{10}$$

where α, β balance the contributions of eigenvalue and eigenvector differences.

To transform spectral distance into a reward, we apply a piecewise linear mapping with a critical threshold τ:

$$R(d) = \begin{cases} R_{\max} - \gamma d, & d \le \tau, \\ R_{\max} - \gamma\tau - \delta(d - \tau), & d > \tau, \end{cases} \tag{11}$$

where γ and δ are decay rates ($\delta > \gamma$), ensuring a gradual reward decline for small errors and sharper penalties for large deviations.

For objects segmented into multiple fragments, we compute the spectral distance and associated reward for each fragment pair individually, then accumulate the total reward as:

$$R_{\text{total}} = \kappa \sum_i R(d_i), \tag{12}$$

with κ a global scaling factor ensuring consistent reward magnitudes across tasks.

This formulation provides an efficient, continuous, and pose-invariant reward signal for evaluating and planning goal-conditioned robotic cutting actions.

In order to evaluate the efficient and pose-invariant property, we conduct the following experiments:

Parameter Definitions and Pose-Invariance Observations – Our spectral-reward pipeline relies on three key configuration parameters and a fixed cutting trajectory. Specifically:

- `num_point` ($= 512$) determines how many points are uniformly sampled from the fragment's point cloud to serve as vertices in the spectral graph. A larger `num_point` yields finer geometric resolution but increases graph-construction cost, whereas a smaller value speeds up computation at the risk of losing subtle shape details.
- k ($= 30$) defines the neighborhood size in the k-nearest-neighbor graph: each sampled point connects to its k closest neighbors (by Euclidean distance). This choice balances local connectivity (capturing fine features) against spectral stability (avoiding overly dense graphs that blur intrinsic geometry).
- *Trajectory* refers to the fixed sequence of knife motions (the "default" cutting action) applied to all shapes. At each discrete timestep along this trajectory, we recompute the fragment's spectral descriptor and record the resulting reward value.

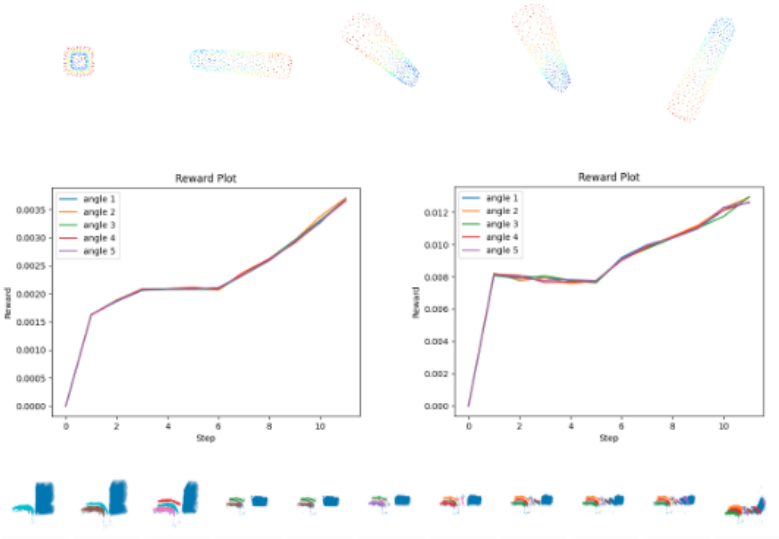

Figure 5: Pose-Invariance Evaluation of the Spectral Reward. Top: Five ideal stick fragments ($5 \times 5 \times 32$) rotated through distinct angles. **Middle:** Reward-versus-cut-step curves for each rotation, computed using dense particles (left) and mesh-surface particles (right) with `num_point`=512 and $k = 30$. **Bottom:** Corresponding fragment point-cloud sequences along the default cutting trajectory. The nearly identical curves across all rotations confirm that our spectral reward is invariant to rigid transformations of the ideal shape.

In our experiment shown in Figure 5, we generate five identical "stick" fragments ($5 \times 5 \times 32$) and rotate each by a distinct yaw angle before sampling and evaluation. We then compute two sets of reward-vs.-step curves—one using all dense particles, the other using only mesh-surface particles. As shown in the overlaid plots, all five curves coincide almost exactly for both representations. This confirms that, under our sampling density (`num_point`=512) and graph connectivity ($k = 30$), the spectral reward is effectively invariant to rigid rotations of the ideal shape.

Multi-step Spectral-Loss Evaluation – To verify that our spectral reward correctly identifies the intended fragment at each cutting step, we conducted a pairwise matching experiment on tele-operated MPPI data. After segmenting the object into its constituent pieces, we compute the spectral distance $d_{\text{spec}}(X_{s_i}, X_{p_j})$ between the shape at cutting state s_i and each fragment p_j, for $i, j = 0, \ldots, 4$. Figure 6 visualizes the resulting 4×5 "spectral-loss" matrix: rows index the initial and four successive cut states, columns index the five fragment geometries (shown below). Darker entries (lower values) indicate stronger intrinsic shape similarity. Notice that for every state s_i, the piece with low spectral distance in previous state still remain in small value despite the fact that the piece might fall down due to gravity and there is a new piece with low spectral distance appears. This demonstrates both the pose-invariance and discriminative power of our spectral descriptor, and justifies its use for automatic labeling and qualitative evaluation of MPPI rollouts.

1.8 MPPI for Data Generation

To generate high–quality demonstrations for policy learning we adopt the *Model Predictive Path Integral* (MPPI) sampling–based optimal control scheme [1]. MPPI maintains a horizon–H open–loop action sequence $\mathbf{U}_{0:H-1} \in \mathbb{R}^{H \times m}$ (with $m = 6$ for the knife pose) and repeatedly refines it by importance–sampling noisy roll–outs in the deformable–body simulator. For each iteration we draw K perturbations $\{\boldsymbol{\epsilon}^{(k)}\}_{k=1}^{K} \sim \mathcal{N}(0, \Sigma)$ and integrate the dynamics forward, harvesting the terminal fragments $X^{(k)}$ produced by each sampled cut. The return of a roll–out is the negative spectral discrepancy

$$J^{(k)} \;=\; -\,R_{\text{spec}}(X^{(k)}, g) - \sum_{t=0}^{H-1} C(o_t^{(k)}, a_t^{(k)}),$$

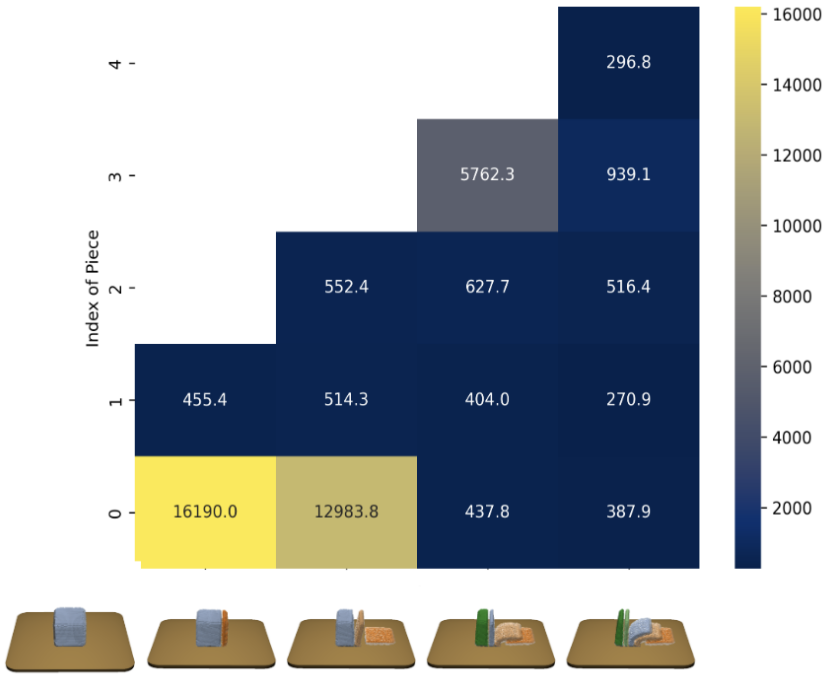

Figure 6: Pairwise spectral-loss matrix between each cutting state (rows) and each piece within the state (columns). Lower values indicate closer intrinsic geometry matches, enabling qualitative MPPI data collection.

where R_{spec} is the pose-invariant spectral reward defined previously, and $C(\cdot)$ is a small quadratic control penalty that biases the knife towards smooth, energy–efficient motions.

Given the roll–out returns we form importance weights $w_k = \exp(-J^{(k)}/\lambda)$ with temperature $\lambda = 1/\kappa$ and update the mean action sequence in closed form:

$$\mathbf{U}_{0:H-1} \leftarrow \mathbf{U}_{0:H-1} + \frac{\sum_{k=1}^{K} w_k \, \boldsymbol{\epsilon}^{(k)}}{\sum_{k=1}^{K} w_k}.$$

After n refinement iterations the first action $\mathbf{U}_0$ is executed on the robot; the horizon is then shifted and replanning continues until the episode terminates.

During each MPPI episode we log the full state–action–reward tuples $\{o_t, a_t, R_{\text{spec}}(X_t, g)\}_{t=0}^{T}$ as well as intermediate mesh reconstructions and knife trajectories. We repeat the procedure for a diverse set of goal shapes and material parameters, producing $\sim 6{,}000$ labelled cuts that serve as expert demonstrations for subsequent behaviour–cloning.

To complement the automatically generated dataset we developed a light-weight tele-operation interface in which the operator steers the knife with a 6-DoF mouse while discrete cutting commands are triggered via keyboard hot-keys. All tele-operated actions are replayed in simulation to obtain exact particle damage labels and the same spectral reward used by MPPI, ensuring perfect consistency between human and algorithmic demonstrations.

Together, MPPI planning guided by spectral rewards and targeted tele-operation yield a rich, high-fidelity dataset that underpins robust policy learning across the wide range of scenarios posed by *TopoCut*.

1.9 Policy Learning in Details

1.9.1 Dynamics-Informed Perception Module

To efficiently encode dense deformable object states for policy learning, we develop a dynamic topology prediction model, denoted as $\mathcal{F}$, which predicts the future topological state conditioned on

the current topological state and action. Formally, we model the dynamics as:

$$\mathcal{F} : (\text{topo}_t, a_t) \mapsto \text{topo}_{t+1}, \tag{13}$$

where both topo_t and a_t are represented as point clouds:

$$\text{topo}_t = \{(\mathbf{x}_i, \mathbf{f}_i)\}_{i=1}^{N}, \quad a_t = \{(\mathbf{x}_j, \mathbf{g}_j)\}_{j=1}^{N}, \tag{14}$$

with $\mathbf{x}_i, \mathbf{x}_j \in \mathbb{R}^3$ denoting 3D coordinates and $\mathbf{f}_i$, $\mathbf{g}_j$ denoting associated point features: cluster labels for topology and binary segmentation masks for action, respectively.

Within $\mathcal{F}$, we introduce a perception encoder Φ that processes point clouds by jointly embedding the 3D coordinates and their corresponding features:

$$\mathbf{z}_t = \Phi(\mathbf{X}_t, \mathbf{F}_t), \tag{15}$$

where $\mathbf{X}_t$ and $\mathbf{F}_t$ are the stacked xyz coordinates and feature vectors, respectively. Φ produces a point-wise embedding $\mathbf{z}_t$ that captures spatial geometry, topological structure, and action intent. This embedding is later reused for downstream goal-conditioned policy learning.

Topological Representation. – For topo_t, the point features $\mathbf{f}_i$ encode the cluster membership obtained from our particle-based damage tracking, represented as a one-hot vector over a maximum of 32 clusters. For a_t, the point features $\mathbf{g}_j$ are binary one-hot vectors indicating whether the point lies on the cutting surface (cut = 1, not cut = 0).

Learning Objective and Intuition. – The dynamic topology model $\mathcal{F}$ is trained to predict how the object's topology evolves under a given action. Specifically, given (topo_t, a_t), it predicts the resulting topological configuration topo_{t+1}. By doing so, Φ learns rich, actionable particle-wise embeddings that encode both geometric and topological transformations, enabling robust downstream manipulation.

Preprocessing and Graph Construction. – Assuming access only to noisy surface observations, we reconstruct the object's mesh using Marching Cubes (Warp [**?**]), sample interior points to form volumetric point clouds, and downsample them using Farthest Point Sampling (FPS). A topology-aware graph is then constructed, where edges connect nearest neighbors within the same cluster, promoting learning of local topological consistency.

Model Architecture. – The perception encoder Φ consists of two stages: (i) a Graph Convolutional Network (GCN) to extract local features from the topology-aware graph, and (ii) a Graph Transformer that globally refines the features via self-attention. The output is a set of point-wise embeddings encoding spatial and topological context.

Training Loss. – We jointly optimize Φ with two complementary objectives: geometric consistency and topological structure prediction. The overall loss is defined as:

$$\mathcal{L} = \lambda_{\text{pos}}\mathcal{L}_{\text{pos}} + \lambda_{\text{topo}}\mathcal{L}_{\text{topo}}, \tag{16}$$

where λ_{pos} and λ_{topo} are balancing weights.

For geometric consistency, we minimize the Chamfer Distance (CD), Earth Mover's Distance (EMD), and Hausdorff Distance(HD) between predicted and ground-truth point clouds:

$$\mathcal{L}_{\text{pos}} = \text{CD}(\hat{X}, X) + \text{EMD}(\hat{X}, X) + \text{HD}(\hat{X}, X) \tag{17}$$

where $\hat{X}$ and X are predicted and ground-truth point sets.

For topological structure prediction, we treat clustering as a contrastive learning task, where only the relative membership between points matters. A Hungarian matching-based loss is applied:

$$\mathcal{L}_{\text{topo}} = \min_{\pi \in \mathcal{P}} \sum_{i,j} \ell(\pi(c_i), \hat{c}_j), \tag{18}$$

where $\mathcal{P}$ is the set of bipartite matchings, c_i and $\hat{c}_j$ are the ground-truth and predicted cluster assignments, and $\ell(\cdot, \cdot)$ is a binary cross-entropy loss.

Training Data. – The dynamic topology model $\mathcal{F}$ is pretrained using state-action-state triplets $(\text{topo}_t, a_t, \text{topo}_{t+1})$ collected from the MPPI-based demonstration generation (Section **??**). This rich dataset covers diverse materials, object geometries, and cutting scenarios, enabling the learned embeddings to generalize effectively across a wide range of deformable manipulation tasks.

1.9.2 Particle-based Score-Entropy Discrete Diffusion Policy

With the pretrained perception encoder Φ that captures geometric and topological information, we now train a goal-conditioned behavior cloning (BC) policy for robotic cutting. The policy operates over perception embeddings, taking as input the current object topology and the desired goal shape, and predicting the next cutting action.

Given a point cloud observation $o_t = (\mathbf{X}_t, \mathbf{F}_t)$ representing the current object state, and a goal point cloud $g = (\mathbf{X}_g, \mathbf{F}_g)$, we first obtain their embeddings:

$$\mathbf{z}_t = \Phi(\mathbf{X}_t, \mathbf{F}_t), \quad \mathbf{z}_g = \Phi(\mathbf{X}_g, \mathbf{F}_g), \tag{19}$$

where $\mathbf{z}_t$ and $\mathbf{z}_g$ are point-wise embeddings encoding geometry and topology.

The cutting action is represented as a binary segmentation $\mathbf{a}_t \in \{0, 1\}^N$, where each point is classified as cut (1) or not cut (0).

Policy Model: Conditional Score-Based Discrete Diffusion. – Inspired by Score Entropy Discrete Diffusion (SEDD) [2], we formulate action prediction as a conditional discrete denoising diffusion process over point-wise binary labels.

In the forward process, the clean action labels $\mathbf{a}_t^*$ are progressively corrupted into noisy labels $\tilde{\mathbf{a}}_t$ through a multinomial noise distribution:

$$q_t(\tilde{\mathbf{a}}_t \mid \mathbf{a}_t^*) = \text{Multinomial}\left(\tilde{\mathbf{a}}_t \mid \mathbf{p}_t(\mathbf{a}_t^*)\right), \tag{20}$$

where $\mathbf{p}_t(\mathbf{a}_t^*)$ denotes the noise schedule at timestep t.

The policy network s_θ is trained to predict the score function:

$$s_\theta(\tilde{\mathbf{a}}_t, t, \mathbf{z}_t, \mathbf{z}_g) \approx \nabla_{\tilde{\mathbf{a}}_t} \log q_t(\mathbf{a}_t^* \mid \tilde{\mathbf{a}}_t), \tag{21}$$

where s_θ outputs the gradient of the log-posterior of the clean action given the noised action, conditioned on the current and goal embeddings.

This formulation enables the policy to iteratively denoise $\tilde{\mathbf{a}}_t$ toward recovering the target cutting action.

Training Objective. – The BC policy is trained by minimizing the Denoising Score Entropy (DSE) loss across all diffusion steps:

$$\mathcal{L}_{\text{BC}}(\theta) = \mathbb{E}_{(o_t, g, \mathbf{a}_t^*) \sim D}\left[\sum_{t=1}^{T} \mathcal{L}_{\text{DSE}}(\tilde{\mathbf{a}}_t, \mathbf{a}_t^*; \theta)\right], \tag{22}$$

where the per-step DSE loss is defined as:

$$\mathcal{L}_{\text{DSE}}(\tilde{\mathbf{a}}_t, \mathbf{a}_t^*; \theta) = \mathbb{E}\left[\left\| s_\theta(\tilde{\mathbf{a}}_t, t, \mathbf{z}_t, \mathbf{z}_g) - \nabla_{\tilde{\mathbf{a}}_t} \log q_t(\mathbf{a}_t^* \mid \tilde{\mathbf{a}}_t) \right\|_2^2\right]. \tag{23}$$

Action Reconstruction. – After completing the denoising sampling process, we obtain a binary segmentation over the points indicating which regions should be cut. From the points classified as cut, we fit a cutting plane, thereby reconstructing the knife's pose and producing the final action a_t.

1. Inputs, Goals, and Perception Embeddings – At each decision step t, the robot observes:

$$o_t = (\mathbf{X}_t, \mathbf{F}_t),$$

where:

- $\mathbf{X}_t = [\mathbf{x}_{t,1}, \ldots, \mathbf{x}_{t,N}]^\top \in \mathbb{R}^{N \times 3}$ are the 3D coordinates of N particles representing the object.

- $\mathbf{F}_t = [\mathbf{f}_{t,1}, \ldots, \mathbf{f}_{t,N}]^\top \in \mathbb{R}^{N \times f}$ are associated per-point features, such as normals, damage indicators, or material properties.

A desired goal shape is provided as another point-cloud:

$$g = (\mathbf{X}_g, \mathbf{F}_g) \in \mathbb{R}^{N \times 3} \times \mathbb{R}^{N \times f}.$$

Both o_t and g are passed through a pretrained perception encoder

$$\Phi : (\mathbb{R}^{N \times 3}, \mathbb{R}^{N \times f}) \longrightarrow \mathbb{R}^{N \times d},$$

which uses graph-based convolutions followed by a small MLP to produce point-wise latent embeddings:

$$\mathbf{z}_t = \Phi(\mathbf{X}_t, \mathbf{F}_t), \quad \mathbf{z}_g = \Phi(\mathbf{X}_g, \mathbf{F}_g),$$

where d is the embedding dimension. Intuitively, $\mathbf{z}_t$ captures the current object's geometric and topological state, while $\mathbf{z}_g$ encodes the desired target shape.

2. Action Representation and Forward Noising – We represent the cutting action at time t as a binary mask

$$\mathbf{a}_t^* = [a_{t,1}^*, \ldots, a_{t,N}^*]^\top \in \{0,1\}^N,$$

where $a_{t,i}^* = 1$ indicates that particle i should be cut. To train our policy using denoising diffusion, we first define a forward noising process that corrupts $\mathbf{a}_t^*$ into progressively noisier versions $\tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_T$. Starting from $\tilde{\mathbf{a}}_0 = \mathbf{a}_t^*$, each step applies a small random flip:

$$q_t(\tilde{\mathbf{a}}_t \mid \tilde{\mathbf{a}}_{t-1}) = \prod_{i=1}^{N} \Big[(1 - \beta_t)\, \delta(\tilde{a}_{t,i} = \tilde{a}_{t-1,i}) \;+\; \beta_t\, \delta(\tilde{a}_{t,i} \neq \tilde{a}_{t-1,i}) \Big],$$

where $\beta_t \in [0,1]$ is a noise-rate schedule (e.g. $\beta_t = t/T$). After T steps, $\tilde{\mathbf{a}}_T$ is nearly uniform random.

3. Score Network and Reverse Diffusion – We train a neural network $s_\theta(\tilde{\mathbf{a}}_t,\, t,\, \mathbf{z}_t,\, \mathbf{z}_g)$ to predict the gradient of the log-probability of the true mask given the noisy mask:

$$s_\theta(\cdot) \;\approx\; \nabla_{\tilde{\mathbf{a}}_t} \;\log\; q_t(\mathbf{a}_t^* \mid \tilde{\mathbf{a}}_t).$$

Concretely, s_θ receives as input:

- The noisy mask $\tilde{\mathbf{a}}_t$ (embedded as scalars).
- The timestep t, encoded with sinusoidal features.
- The concatenated perception embeddings $[\mathbf{z}_t; \mathbf{z}_g] \in \mathbb{R}^{N \times 2d}$.

During inference, we reverse the noising chain by sampling

$$p_\theta(\mathbf{a}_{t-1} \mid \tilde{\mathbf{a}}_t, t, \mathbf{z}_t, \mathbf{z}_g) = \mathrm{Categorical}\Big(\mathrm{softmax}(\log(1 - \beta_t,\, \beta_t) + s_\theta(\tilde{\mathbf{a}}_t, t, \mathbf{z}_t, \mathbf{z}_g))\Big).$$

This step "denoises" $\tilde{\mathbf{a}}_t$ one level at a time back toward a clean mask.

4. Training Objective: Denoising Score-Entropy Loss – We optimize θ by minimizing the expected squared error between the network's predicted score and the true score over all timesteps:

$$\mathcal{L}_{\mathrm{BC}}(\theta) = \mathbb{E}_{(o_t, g, \mathbf{a}_t^*) \sim D} \left[\sum_{t=1}^{T} \underbrace{\mathbb{E}_{\tilde{\mathbf{a}}_t \sim q_t(\cdot \mid \mathbf{a}_t^*)} \| \, s_\theta(\tilde{\mathbf{a}}_t, t, \mathbf{z}_t, \mathbf{z}_g) \;-\; \nabla_{\tilde{\mathbf{a}}_t} \;\log\; q_t(\mathbf{a}_t^* \mid \tilde{\mathbf{a}}_t) \|_2^2}_{\mathcal{L}_{\mathrm{DSE}}(\tilde{\mathbf{a}}_t, \mathbf{a}_t^*; t)} \right].$$

Because q_t uses simple bit-flip noise, the true score gradient has a closed form:

$$\nabla_{\tilde{a}_{t,i}} \log q_t(a_i^* \mid \tilde{a}_{t,i}) = \frac{\delta(\tilde{a}_{t,i} = a_i^*) - (1 - \beta_t)}{\beta_t (1 - \beta_t)}.$$

5. Inference and Action Reconstruction – At test time:

1. Initialize $\tilde{\mathbf{a}}_T$ by sampling each bit from $\mathrm{Bernoulli}(0.5)$.

2. For $t = T, T-1, \ldots, 1$, sample $\mathbf{a}_{t-1}$ from $p_\theta(\mathbf{a}_{t-1} \mid \tilde{\mathbf{a}}_t, t, \mathbf{z}_t, \mathbf{z}_g)$.
3. The final mask $\tilde{\mathbf{a}}_0$ indicates the cut points.
4. Fit a planar cut by solving

$$\min_{\mathbf{n},d} \sum_{i:\,\tilde{a}_{0,i}=1} (\mathbf{n}^\top \mathbf{x}_{t,i} + d)^2, \quad \|\mathbf{n}\|_2 = 1,$$

where $\mathbf{n}$ is the plane normal and d its offset.

Summary. – Our BC policy systematically integrates dynamics-informed perception embeddings and conditional score-based discrete diffusion modeling. By treating cutting action prediction as a goal-conditioned denoising process over discrete labels, our method achieves robust and generalizable cutting behavior across diverse object shapes and cutting goals.

PDDP naturally handles the combinatorial nature of cutting (multimodal mask distributions) and provides smooth trade-offs between sample quality and runtime via T. The closed-form score supervision ensures stable training, and the iterative denoising generates crisp, coherent cutting actions that generalize across object shapes and materials.

Overall, our method systematically combines efficient topology tracking, pose-invariant shape evaluation, dynamics-informed perception, and supervised policy learning to enable effective robotic cutting in general scenarios.

References

[1] G. Williams, P. Wagener, B. Goldfain, P. Drews, J. Rehg, and E. Theodorou. Information–theoretic model predictive path integral control: Theory and applications to robotics. In *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[2] A. Lou, C. Meng, and S. Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution, 2024. URL https://arxiv.org/abs/2310.16834.