Appendix

A. Real-World Experiment Details

1) Dataset Collection, Pretraining and Training: For the real-world experiments, we collected the LEAP Hand dataset and trained a model independently. We initially selected 78 daily objects from the YCB dataset [35] and ContactDB [36], then applied the DFC-based [2] grasp optimization method from [37] to generate 1,000 grasps per object, yielding a total of 78,000 grasps. Following a dataset filtering process, we obtained 24,656 grasps across 73 objects. The encoder network was first pretrained on the original dataset, and the entire model was then trained on the filtered dataset, as described in Sec. III.

2) Real-World Deployment Details: We first scanned the objects listed in Tab. IV using AR Code [38]. After camera intrinsics [39] and extrinsics [40] calibration, we estimated object poses using FoundationPose [41] and sampled point cloud uniformly on their surfaces. In this tabletop grasping setting, only top-down and side grasps are feasible, as other palm orientations would likely collide with the table. To address this, the model took as input the sampled object point clouds and a batch of LEAP Hand point clouds, which corresponded to 32 interpolated hand poses ranging from top-down to right-side orientations, enabled by our palm orientation control functionality. We randomly selected one of the top-5 grasps from the generated batch, ranked according to the same grasp energy calculation used during dataset generation [37]. We then use MPLib [42] for arm motion planning to the desired end-effector pose. A PD controller is applied for grasp execution.

3) Experiment Result: We tested 10 objects with various shapes, performing 10 grasping attempts for each object. The experimental results are shown in Tab. IV and Fig. 8. Our method achieved an average success rate of **89%** across these 10 objects, demonstrating the effectiveness of our method in dexterous grasping and its generalizability to novel objects.

Apple	**Bag**	**Brush**	**Cookie Box**	**Cube**
9/10	10/10	9/10	10/10	9/10
Cup	**Dinosaur**	**Duck**	**Tea Box**	**Toilet Cleaner**
7/10	9/10	8/10	8/10	10/10

TABLE IV: Real-world experiment results on unseen objects.

B. Zero-shot Generalization to Novel Hands Experiment

We trained the model separately on each of the three robotic hands and then validated it on the others without further training. As shown in Tab. V, the results indicate that when transferring from high-DOF hands to low-DOF hands in a zero-shot setting, the model retains a certain level of performance. However, transferring in the opposite direction largely fails. We hypothesize that this difference arises because high-DOF hands have a much more complex configuration space, allowing the model to learn a broader range of articulation-invariant matching tasks, which can still perform well on the simpler articulation-invariant tasks required for low-DOF hands. In contrast, the configuration space of low-DOF hands is relatively simple, and when trained on these hands, the model can only master simple articulation-invariant matching tasks.

Training Robot	**Success Rate (%) ↑**		
	Allegro	Barrett	ShadowHand
Allegro	(88.70)	83.60	1.10
Barrett	42.40	(84.80)	6.90
Shadowhand	56.90	83.70	(75.80)

TABLE V: Generalization results to novel hands.

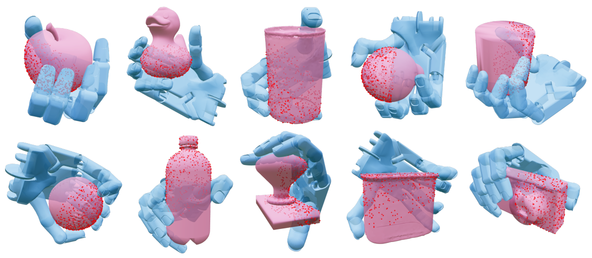


Fig. 9: Grasp examples with partial object point clouds. Red points show the observed portion.

C. Partial Object Point Cloud Sampling

Given the mesh of an object, we begin by randomly sampling $2 \times N_O$ points. Next, a point is randomly sampled on a unit sphere, and the direction vector r from this point to the origin is computed. For each point in the point cloud, we calculate the dot product between r and the corresponding direction vectors d_i. We then remove half of the points with the smallest dot product values $r \cdot d_i$, leaving a subset of N_O points, which forms the partial object point cloud. This process is used to generate random point clouds during both training and evaluation.

D. Grasp Controller

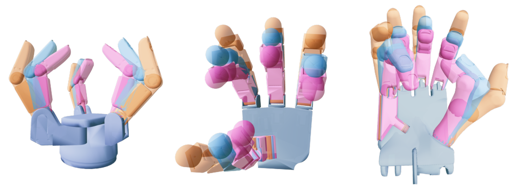
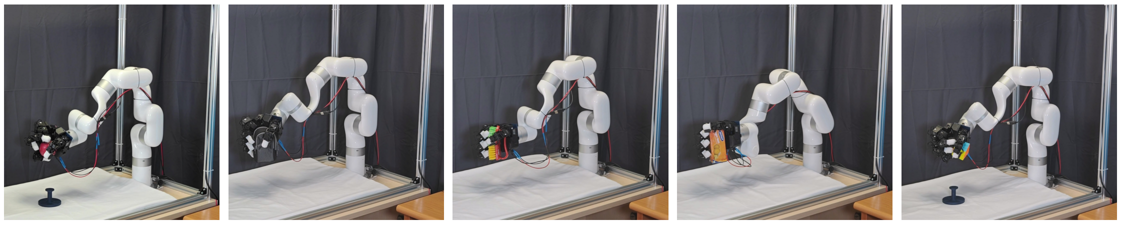
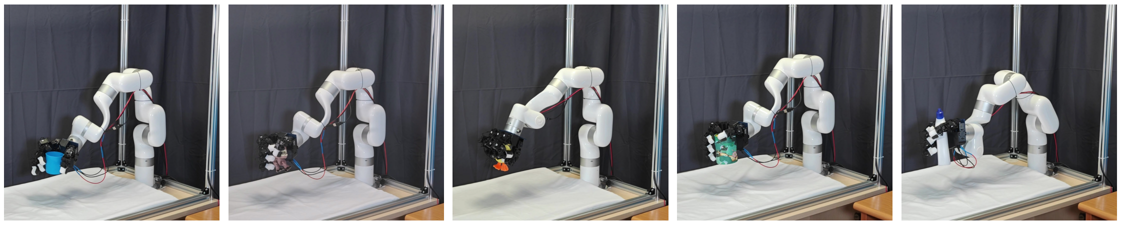


Fig. 10: Visualization of the grasp controller's effect: blue indicates the predicted grasp pose, orange represents q_{outer}, and pink represents q_{inner}.

To mitigate minor inaccuracies and subtle penetrations commonly found in generative methods, as well as the limitations of directly predicting a static grasp pose—which overlooks the forces exerted on contact surfaces—we developed a heuristic grasp controller to better simulate real-world grasping scenarios. The controller aims to generate a



(a) Apple (b) Bag (c) Brush (d) Cookie Box (e) Cube



(f) Cup (g) Dinosaur (h) Duck (i) Tea Box (j) Toilet Cleaner

Fig. 8: Real-world grasp demonstrations

configuration q_{outer} that is farther from the object's center of mass and a configuration q_{inner} that is closer to the center of mass, based on the predicted pose. Fig. 10 illustrates the impact of the grasp controller.

1) Evaluation Metric Details: In Isaac Gym, we evaluate the success of a grasp through a two-phase process. First, in the grasp phase, we use the previously described grasp controller to compute q_{outer} and q_{inner}. We set the robot joint position to q_{outer} with a position target at q_{inner}. Then we simulate for 1 second, equivalent to 100 simulation steps for the hand to close and grasp. In the second phase, we apply disturbance forces sequentially along six orthogonal directions, following the method in [12]. These forces are defined as:

$$F_{\pm xyz} = 0.5m/s^2 \times m_{\text{object}} \tag{15}$$

where m_{object} denotes the mass of the object.

Our approach improves upon [12] by introducing a dynamic grasp phase, transitioning the evaluation from static to dynamic, and thereby significantly enhancing the rigor of the evaluation metric. In the original static validation, some grasps could hold objects in unstable positions. By introducing dynamic validation, these unstable grasps are less likely to succeed, resulting in a more stringent and accurate assessment of grasp quality. Moreover, static validation is prone to simulation errors, such as object penetration or robot self-collisions, which can incorrectly classify unstable grasps as successful. The dynamic method alleviates these issues, providing a more robust and reliable evaluation of grasp success.

Fig. 11 illustrates several anomalous grasps that, despite appearing to fail, could still be judged as successful under the static metric. These grasps, either in an unstable state or exhibiting significant self-penetration, are impractical for real-world applications, highlighting the limitations of static validation.

2) Dataset Filtering: To address the suboptimal grasp quality, we applied a filtering process to the CMapDataset [12]. Specifically, each grasp in the dataset was evaluated based on the success metrics defined in Sec. IV-A and Appendix D.1. We then store the relative 6D pose and joint values of every successful grasp in the filtered dataset.

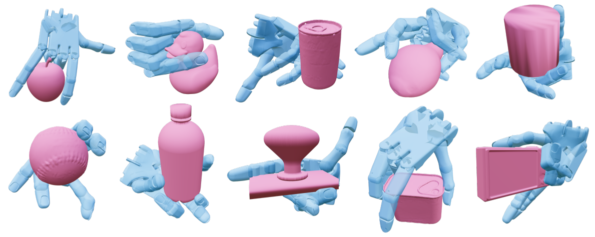


Fig. 11: Grasp examples filtered out from the dataset that would otherwise be deemed successful under static metric.

E. Baseline Description

1) DFC [2]: Since DFC is a purely optimization-based method, the speed of generating grasps is particularly slow. Therefore, we evaluate it using the original CMapDataset, which was primarily generated by the DFC method. As the dataset generation process also minimizes the hand prior energy and penetration energy described in [12], and some generated grasps may have already been filtered, the evaluation results are likely better than DFC's actual performance.

2) GenDexGrasp [12]: We used the filtered grasp dataset to train the model, where the contact heatmap was generated using the aligned distance as described in the paper. The GenDexGrasp model was trained with default hyperparameters. In Tab. VI, we compared the results of the open-source pretrained model with those of our trained model, demonstrating that our filtered dataset is of higher quality.

3) ManiFM [13]: Due to the unavailability of pretrained models for Barrett and ShadowHand, our evaluation was restricted to the Allegro pretrained model. Considering the fundamental differences between point-contact and surface-contact grasps, we optimized the controller's hyperparameters for improved performance of ManiFM. Nevertheless, despite the seemingly high quality of the generated grasps, the

Method	Success Rate (%) ↑			
	Allegro	Barrett	ShadowHand	Avg.
pretrain	51.00	63.80	44.50	53.10
train	51.00	67.00	54.20	57.40

TABLE VI: GenDexGrasp Result Comparision

inherent instability of point-contact grasps posed significant challenges in achieving a high success rate during simulation.

4) GeoMatch [10]: Although GeoMatch is a keypoint matching-based method that supports cross-embodiment and shares similarities with our approach, we faced challenges in reproducing its results due to the absence of pretrained models and insufficient details regarding the data file formats, which remained unsolved in its repository's issues as well. Consequently, it was not included as a baseline for comparison.

F. Network Architecture

1) Point Cloud Encoder: To map robot and object features into a shared feature space, enhancing the network's ability to learn correspondences between them, we employed identical architectures for both the robot and object encoders. Our encoder design is based on the DGCNN [27] architecture, as implemented in [30]. Notably, this implementation omits the original layer-wise re-computation of K-nearest neighbors (KNN) for graph construction, resulting in a "Static Graph CNN". In our setup, K is set to 32, meaning that each point's receptive field is much smaller than the total number of points in the cloud ($N_{\mathcal{R}} = 512$). This constraint limits the ability of the per-point feature extraction process to capture global information, which poses a challenge for the object encoder, as it struggles to learn comprehensive geometric shape features.

We experimented with the original dynamic graph structure, but it led to a decline in pretraining performance. We hypothesize that, for configuration-invariant learning objectives, local structural information in the point cloud is critical, and the network needs to be reinforced to capture this. The dynamic graph structure tends to learn similar structures across different fingers, which, while beneficial for segmentation tasks, is less suited to our specific learning goals. The impact of varying network architectures and feature learning strategies will be further explored in future work.

Consequently, our encoder follows a "Static Graph CNN" architecture with five convolutional layers. After the last convolution, a global average pooling layer generates a global feature concatenated with features from all previous layers. This combined output is passed through a final convolutional layer, projecting into the embedding dimension. The architecture is illustrated in Fig. 12, where the LeakyReLU activation function uses a negative slope of 0.2.

2) Cross-Attention Transformer: We followed the architectural design from [30], utilizing a multi-head attention

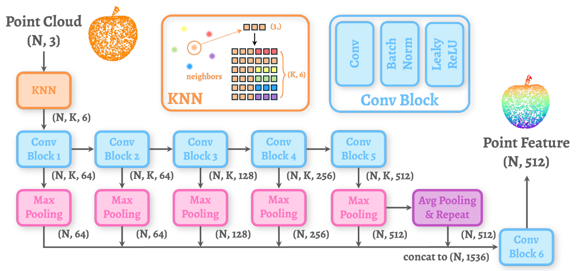


Fig. 12: Point cloud encoder architecture.

block with 4 heads. The implementation details can be found in the code.

3) Kernel MLP: We adopted the same hyperparameters design as [30]. Specifically, the MLP consists of two hidden layers with feature dimensions of 300 and 100, respectively, along with the ReLU activation function.

G. Dataset Preprocessing

1) URDF File Preprocessing: To facilitate optimization, we introduce six virtual joints between the world frame and the robot's root link: three prismatic joints representing translation (x, y, z) and three revolute joints representing rotation $(roll, pitch, yaw)$. These virtual joints are incorporated into the robot's URDF file and treated equivalently to other joints to simplify the computation of the Jacobian matrix. Furthermore, virtual links are added to the distal ends of each tip link to address potential errors in the 6D pose during optimization, ensuring consistent constraints across all links despite reduced rotational restrictions.

2) Robot Point Cloud Sampling: To extract the stored point clouds $\{\mathbf{P}_{\ell_i}\}_{i=1}^{N_\ell}$ from the URDF file of a specific robot, we first sample 512 points from the mesh of each link. We then apply the Farthest Point Sampling (FPS) algorithm to the complete point cloud, selecting 512 points, denoted as N_R in our method. These point clouds are stored separately for each distinct link.

This process guarantees that, for any joint configuration, our point cloud forward kinematics model, FK $\left(q, \{\mathbf{P}_{\ell_i}\}_{i=1}^{N_\ell}\right)$, can map joint configurations to corresponding point clouds at new poses. This ensures consistent point cloud correspondence across different poses, a key advantage for our pretraining methodology.

3) Object Point Cloud Sampling: Starting with the mesh file of an object, we initially sample 65,536 points. For each training iteration, we randomly select 512 points from this set and apply Gaussian noise $\mathcal{N}(0, 0.002)$ for data augmentation. This strategy improves the model's generalization across different object shapes.

H. Matrix Block Computation

To address the high GPU memory demands of using the MLP kernel function to compute $\mathcal{D}(\mathcal{R}, \mathcal{O})$, we implemented a matrix block computation strategy to optimize memory usage. After experimentation, we ultimately chose to divide the entire matrix into 4×4 blocks for computation, which reduces memory consumption by approximately 34% while maintaining similar computation time.