Contact-GraspTransformer | 37

Devansh Royal J., Aida Mirebrahimi, Tim Wang, Dinesh Varun Shankar K.
Carnegie Mellon University
IDL Spring26
{djonnala, amirebra, twang3, dkandiya}@andrew.cmu.edu

Abstract

Robotic grasping in cluttered environments requires reasoning over complex 3D geometry from partial observations. Contact-GraspNet (CGN) addresses this by predicting grasps directly from point clouds, but relies on PointNet++, limiting global context modeling. We propose Contact-GraspTransformer (CGT), which replaces the PointNet++ backbone with Point Transformer v3 (PTv3) to enable attention-based feature aggregation over point clouds. Evaluated on a subset of the ACRONYM dataset, CGT reduces test loss ($0.82 \rightarrow 0.75$) and significantly improves inference time (113.64 ms $\rightarrow$ 66.97 ms, $\Delta = 46.67$ ms), at the cost of increased model size. These results show that transformer-based backbones improve both efficiency and performance in 3D grasp prediction, highlighting a trade-off between global modeling capacity and parameter efficiency.

1 Introduction

Robotic manipulation in unstructured environments requires reasoning over complex 3D geometry from partial and noisy observations. Traditional grasping pipelines rely on intermediate steps such as object segmentation and full 3D reconstruction, which are prone to failure in cluttered or occluded scenes [1, 2]. Errors in these stages propagate through the pipeline, and the assumption of clean object boundaries rarely holds in practice.

Early grasping methods were largely analytic, relying on explicit object geometry and hand-designed quality metrics [1, 10]. More recently, data-driven approaches have enabled learning grasp predictions directly from sensory input. Systems such as Dex-Net 2.0 and GG-CNN demonstrate robust planar grasping from partial observations, but operate in a restricted action space that does not generalize to arbitrary 3D grasp configurations [7, 8].

To address this limitation, subsequent work formulates grasp generation directly in 3D using point cloud representations. Architectures such as PointNet and PointNet++ enable learning on unordered point sets through permutation-invariant operations and local neighborhood aggregation [11, 12]. Building on these, methods such as PointNetGPD, 6-DoF GraspNet, and GraspNet-1Billion establish point-cloud-based 6-DoF grasp estimation as a dominant paradigm [6, 9, 4]. However, grasp prediction in SE(3) remains challenging due to its high dimensionality and multi-modal nature.

Contact-GraspNet (CGN) addresses this challenge by adopting a contact-centric formulation, predicting grasp parameters relative to observed points in the scene rather than regressing unconstrained 6-DoF poses [14]. This enables dense grasp prediction over full point clouds without requiring explicit segmentation. However, CGN relies on PointNet++ for feature extraction, which aggregates information over local neighborhoods. While effective, this limits the ability to capture long-range dependencies and global context, and introduces computational overhead through repeated neighborhood queries.

Point cloud transformers address these limitations by replacing explicit neighborhood construction with attention-based aggregation. Earlier approaches rely on costly KNN operations [17], while Point

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Transformer v3 (PTv3) instead serializes point clouds using space-filling curves and applies attention over structured sequences [15]. This enables efficient feature propagation across both local and global regions.

Motivated by these advances, we investigate whether transformer-based architectures can improve grasp prediction within the CGN framework. Specifically, we implement CGN as a baseline and replace its PointNet++ backbone with PTv3, forming a Contact-GraspTransformer model. Our objectives are (1) to evaluate whether global attention improves grasp prediction performance, and (2) to compare inference efficiency between PointNet++ and PTv3.

Our results show that the transformer-based model achieves improved grasp prediction performance while significantly reducing inference time compared to the PointNet++ baseline ($\Delta = 46.667$ ms). However, this improvement comes with a substantial increase in model size, highlighting a trade-off between representational capacity and parameter efficiency in 3D grasp learning.

2 Methodology

2.1 Problem Formulation and Input Representation

We follow the Contact-GraspNet (CGN) formulation [14], where a 3D scene is represented as an unordered point cloud $P \in \mathbb{R}^{N_0 \times 3}$, and the model predicts grasp parameters for a set of anchor points. In the original CGN implementation, $N_0 = 20{,}000$ input points are used, with predictions made on a reduced set of $N = 2{,}048$ anchor points.

In our implementation, we standardize both the baseline (PointNet++ [12]) and proposed (PTv3 [15]) models to operate on $N_0 = 4{,}096$ input points and predict grasps at $N = 4{,}096$ points. This modification ensures a fair comparison between architectures while significantly reducing computational cost and enabling faster iteration during training and ablation studies. Given our focus on simplified scenes with fewer objects, this point density is sufficient to capture the relevant geometric structure.

Point cloud processing architectures must be permutation invariant, ensuring that predictions do not depend on input ordering. Traditional approaches such as PointNet++ [12] achieve this through Farthest Point Sampling (FPS) for downsampling and k-Nearest Neighbors (k-NN) for local feature aggregation. While effective, these operations introduce significant computational overhead and limit the receptive field to local neighborhoods, especially as point cloud density increases.

To address these limitations, we replace the PointNet++ backbone with a transformer-based architecture (PTv3 [15]), which eliminates explicit neighborhood construction and enables efficient global feature aggregation. This allows the model to scale more effectively while capturing both local and long-range geometric dependencies.

2.2 Proposed backbone: Point Transformer v3 (PTV3)

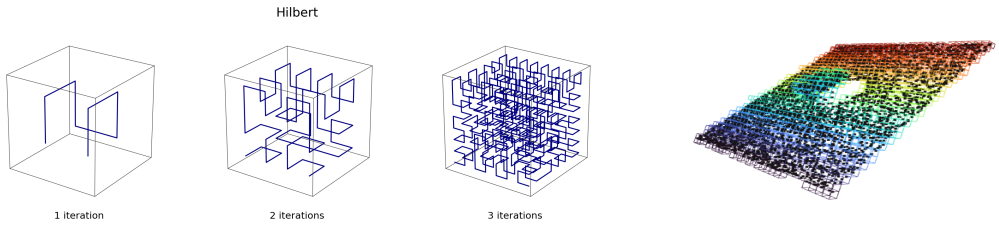
We extend the Contact-GraspNet (CGN) [14] framework by replacing the PointNet++ [12] backbone with a Point Transformer v3 (PTv3) [15] architecture, while preserving the original contact-based prediction heads. This results in a Contact-GraspTransformer (CGT) model that maintains the CGN formulation but enhances its feature representation. At a high level, the model follows an encoder-decoder structure in which the input point cloud is processed by a hierarchical transformer backbone to produce per-point feature embeddings. These embeddings are then used to predict grasp confidence, orientation, and gripper width for each point. The key distinction from the baseline lies in the backbone: CGT replaces local neighborhood aggregation with attention-based feature propagation, enabling the model to capture both local geometry and global context more effectively. The following subsections describe the main components of this architecture.

2.2.1 Voxelization and Serialization

The raw point cloud $\mathbf{P} \in \mathbb{R}^{B \times N \times 3}$ is quantized by a grid size g_{size} to assign each point discrete, integer grid coordinates. During quantization, multiple points may share the same coordinate but remain distinct entities within the (B, N, C) feature tensor. These points, indexed by their spatial grid coordinates, serve as input tokens for the hierarchical transformer stages. Standard transformer architectures are permutation-invariant and require explicit positional cues to understand spatial

structure. PTv3 [15] utilizes space-filling curves (SFCs) to project 3D coordinates into a 1D serialized sequence, ensuring that points proximal in 3D space remain generally proximal within the 1D input. To mitigate spatial discontinuities inherent in any single curve, PTv3 implements a multi-pattern serialization strategy utilizing four distinct orderings:

- Morton (Z-order)
- Transposed Morton (TZ)
- 3-D Hilbert
- Transposed 3-D Hilbert



(a) 3-D Hilbert curve. (b) Voxelization.

Figure 1: Point serialization (left) and voxelization (right).

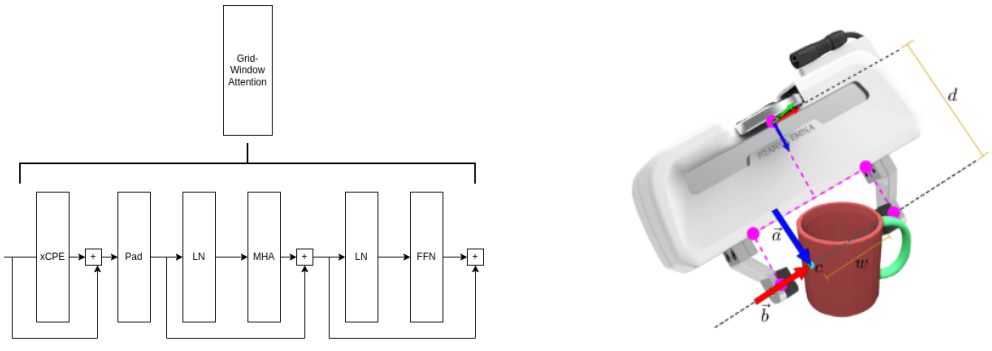
See Appendix A for side-by-side visualizations of the four space-filling curves used for serialization.

Each block within an encoder/decoder stage utilizes a different ordering from this set. Furthermore, a "Shuffle Order" mechanism is applied during training, where the sequence of these four patterns is randomly permuted for every forward pass. This prevents the model from overfitting to a specific serialization sequence and encourages the learning of robust geometric features.

2.2.2 Encoder/Decoder Stage

The PTv3 backbone follows a hierarchical encoder-decoder design composed of stacked Grid-Window Attention blocks (Figure 2a). Within each block, the serialized point sequence is processed through a sequence of operations including contextual positional encoding (xCPE), layer normalization (LN), windowed multi-head attention (MHA), and a feed-forward network (FFN), with residual connections applied after each major component.

Attention is computed within fixed-size windows over the serialized sequence, while different serialization patterns across layers allow information to propagate globally. Encoder stages progressively downsample the representation through pooling, increasing the receptive field, while decoder stages restore resolution via unpooling and skip connections. This structure enables efficient aggregation of both local and global features while preserving spatial detail for grasp prediction.



(a) Grid-Window Attention layers. (b) Annotated Franka Gripper.

Figure 2: Comparison of the PTv3 encoder stage architecture (left) and the physical gripper geometry used for contact anchoring (right).

2.2.3 Explicit Conditional Positional Encoding (xCPE)

The PTv3 backbone combines multi-head attention (MHA) with Explicit Conditional Positional Encoding (xCPE) to process point cloud features. Because point clouds are serialized into 1D sequences, explicit spatial structure is not directly preserved; xCPE is therefore applied as a residual component before each attention block (Figure 2a) to inject local 3D context into the features. This encoding is computed using a lightweight operation over nearby points, allowing the model to capture local geometric information without explicit neighborhood queries. The resulting features are then processed by attention layers, enabling the model to incorporate both local spatial context and broader feature interactions.

2.2.4 CGN Heads

The output features from the final decoder stage ($\mathbf{F} \in \mathbb{R}^{B\times 32\times N}$) are processed by four prediction branches to estimate per-point grasp candidates. Each branch uses 1×1 convolutions to preserve point-wise correspondence while mapping features to grasp parameters.

- **Confidence head:** predicts one logit per point (probability of being a grasp center).
- **Orientation head ($\mathbf{z}_1$):** predicts a 3D vector $\mathbf{z}_1$.
- **Orientation head ($\mathbf{z}_2$):** predicts a 3D vector $\mathbf{z}_2$.
- **Width head:** predicts a distribution over 10 width bins spanning $[0, 0.08]$ m.

The two orientation vectors are converted into a valid 3D frame via an in-network Gram–Schmidt step, producing the baseline ($\mathbf{b}$) and approach ($\mathbf{a}$) directions, which are highlighted in 2b.

Table 1: Architecture of CGN prediction heads.

Head / component	Layers	Output / hyper-parameters
Confidence	Conv1d(32, 64) → BN → ReLU → Conv1d(64, 1)	1 logit / point
Orientation ($\mathbf{z}_1$)	Conv1d(32, 64) → BN → ReLU → Conv1d(64, 3)	$\mathbf{z}_1 \in \mathbb{R}^3$
Orientation ($\mathbf{z}_2$)	Conv1d(32, 64) → BN → ReLU → Conv1d(64, 3)	$\mathbf{z}_2 \in \mathbb{R}^3$
Orthonormalization	Gram–Schmidt	$\mathbf{b}, \mathbf{a}$ (unit vectors)
Width	Conv1d(32, 64) → BN → ReLU → Conv1d(64, 10)	10 bins ($w_{\max} = 0.08$ m)

Table 2: Architecture and training hyper-parameters for the PTv3 backbone.

Model / stage	Hidden layers	Parameters / hyper-parameters
Stem	Linear → BN → GELU	in_channels = 3, out_channels = 32
Encoder Stage 0	2 × {xCPE, LN, MHA, LN, MLP(4x)}	C = 32, Heads = 4, Window = 256
Down 0 (Pool)	Linear, Morton scatter-mean, BN, GELU	32 → 64, pool_stride = 2
Encoder Stage 1	2 × {xCPE, LN, MHA, LN, MLP(4x)}	C = 64, Heads = 4, Window = 256
Down 1 (Pool)	Linear, Morton scatter-mean, BN, GELU	64 → 128, pool_stride = 2
Encoder Stage 2	2 × {xCPE, LN, MHA, LN, MLP(4x)}	C = 128, Heads = 8, Window = 256
Down 2 (Pool)	Linear, Morton scatter-mean, BN, GELU	128 → 256, pool_stride = 2
Encoder Stage 3	2 × {xCPE, LN, MHA, LN, MLP(4x)}	C = 256, Heads = 8, Window = 256
Up 2 (Unpool)	Linear(coarse) + Linear(skip), gather, BN, GELU	256 → 128, skip = 128
Decoder Stage 2	2 × {xCPE, LN, MHA, LN, MLP(4x)}	C = 128, Heads = 8, Window = 256
Up 1 (Unpool)	Linear(coarse) + Linear(skip), gather, BN, GELU	128 → 64, skip = 64
Decoder Stage 1	2 × {xCPE, LN, MHA, LN, MLP(4x)}	C = 64, Heads = 4, Window = 256
Up 0 (Unpool)	Linear(coarse) + Linear(skip), gather, BN, GELU	64 → 32, skip = 32
Decoder Stage 0	2 × {xCPE, LN, MHA, LN, MLP(4x)}	C = 32, Heads = 4, Window = 256
Output Proj.	Linear mapping and tensor transpose	32 → 64

2.3 Dataset

We use a subset of the ACRONYM dataset [3], which provides annotated 6-DoF grasp configurations for objects from ShapeNetSem [13]. In the original pipeline, cluttered scenes are generated in a

physics simulator, rendered with an RGB-D camera, and converted into point clouds with associated ground-truth grasp labels.

In our implementation, we pre-generate these scenes and store them as `.npz` files containing point clouds and corresponding grasp annotations. Each input consists of a randomly sampled point cloud of $N_0 = 4{,}096$ points. Additional data variation is introduced by modifying camera extrinsics, i.e., its position and orientation relative to the scene.

To enable efficient training, we use a subset of the dataset consisting of 15 object categories with 12 objects per category (180 total). Each object is observed from 5–10 viewpoints. For each category, 10 objects are used for training and validation, with validation performed on unseen viewpoints, while the remaining 2 objects are held out for testing to evaluate generalization to unseen objects.

2.4 Evaluation Metric

CGN's ultimate goal is to provide an end-to-end pipeline for grasping classification via scene input in point cloud form to stable grasping. While the most reliable evaluation would be simulating the grasp with a physical robot, oftentimes the real-to-sim translation for the pipeline is non-trivial. In our case, we resort to using the formulated loss value described later to be our overarching evaluation metric. For graphical understanding of these grasps, we can observe the image output in which potential grasps have been annotated onto the image itself. We can then evaluate the images and their grasps by their canonical looks.

Additionally, we evaluate the predicted grasps beyond the loss function by simulating them in a physics simulator, covered in appendix C. In this situation, sim evaluation eliminates the possibility for external perturbation as well as a more vacuumed environment in which we can evaluate the grasps. By defining the trajectory in which a simulated robot can grasp the object, we can see how well the object can be wielded. This can be done by performing future motions with the robot while the pinch is still active and seeing if it is stable.

The baseline utilizes a Contact-Based Grasp Representation, which simplifies the complex task of 6-DOF pose estimation by anchoring grasps to observed surface geometry. Instead of regressing arbitrary poses in unconstrained $SE(3)$ space, the model predicts grasp parameters relative to a specific contact point $c \in \mathbb{R}^3$ identified on the object's observable surface. The grasp pose $g \in G$ is defined by the tuple (R_g, t_g), where:

- Translation (t_g): Calculated as $t_g = c + \frac{w}{2}b + da$. This positions the gripper relative to the contact point c, the predicted grasp width w, and a fixed hardware offset d.
- Rotation (R_g): Constructed as an orthogonal matrix $R_g = [b, a \times b, a]$, where a is the approach vector and b is the baseline vector.

In the ACRONYM dataset, there are 2000 grasps per object. While we train, we are trying to match the current view of the object to one of the viable supervised grasps. It is completely possible for our model to predict new viable grasps, but this would require some other metric to evaluate how stable this grasp is. We know that the ACRONYM's ground truth grasp labels are secure graps under perturbation. Let G_i represent the center point of the gripper of the ith ground truth grasp and $\hat{G}$ represent our the center point of the gripper of our model's predicted grasp. Note that the pink point in Fig. 1 as the center point of the gripper. Our evaluation metric is formulated as such:

$$\min_i \|G_i - \hat{G}\|$$

As detailed in the original paper as well, we denote success and failure cases. If the predicted grasp is beyond 5mm away from any ground truth grasp, we label it as a failed grasp.

2.5 Loss Function

The loss function for CGN is composed of three parts to make the overall loss:

$$l = \alpha l_{bce,k} + \beta l_{add-s} + \gamma l_{width}$$

The first term is defined as the binary cross entropy loss between the predicted output points and the ground truth points. These points define a baseline for where potential grasping is enabled. This term is only calculated for the top k values with the largest error.

The second term is the grasping quality of the gripper itself. It provides a 6-DOF grasping configuration for the gripper to generate a stable grasp on the object at hand. Using five points on the gripper (Fig. 2b), the following transforms are applied to generate the final loss:

$$v_i^{gt} = vR_{g,i}^T + t_{g,i}, \quad v_i^{pred} = v\hat{R}_{g,i}^T + \hat{t}_{g,i}$$

These values are then put into a minimum average distance metric to finalize grasping loss:

$$l_{add-s} = \frac{1}{n^+} \sum_i^{n^+} \hat{s_i} \min ||v_i^{pred} - g_u^{gt}||_2,$$

where n^+ defines the number of feasible contact points.

The final term is the grasp width of the gripper. If a gripper is able to reduce its width it would have a much tighter grasp of the target object. Thus, this term is also added into the total loss formula.

All these terms are weighted by respective coefficients α, β, γ respectively. CGN defines these terms to be $\alpha = 1, \beta = 10, \gamma = 1$ for their purposes, but modified them through a hyperparameter study.

3 Baseline and Extensions

3.1 Baseline : PointNet++

The baseline uses an asymmetric PointNet++ encoder-decoder. The encoder uses Set Abstraction (SA) layers to perform downsampling via Farthest Point Sampling (FPS) and local feature extraction using MLPs. For an input of N points, the output f_j is defined as:

$$f_j = \max_{x_i \in \mathcal{N}(x_j)} \{MLP([x_i - x_j, f_i])\} \tag{1}$$

The decoder employs Feature Propagation (FP) layers with inverse-distance weighting to interpolate features from sparse to dense points. Our inital search for finding a baseline came from top down grasping such as Dex-Net and GG-CNN. While 2D-based methods like Dex-Net offer high throughput, they often struggle with depth ambiguities and the 'thin-feature' problem in complex SE(3) space. By adopting Contact Grasp Net (CGN), we leverage local geometric features through the PointNet++ backbone, which allows for more precise grasp sampling in unstructured 3D environments compared to top-down approaches.

Our baseline implementation is the vanilla CGN architecture trained with our subsetted ACRONYM dataset. We selected this model because it is a well accreddited grasping baseline that has evidence of working well with the dataset of our choice.

This means that we use the CGN embedding and decoder as well as the native backbone PointNet++. We use a V100 GPU as the hardware in which we train the baseline on and train for 180 epochs.

We adhered to the following vanilla hyperparameter configuration to ensure a fair baseline:Optimizer: Adam optimizer with an initial learning rate of 10^{-3}. **Input Processing:** Point clouds were sampled at a density of $4,096$ points per scene, sub-sampled to $4,096$ points for the Set Abstraction layers. **Training Schedule:** We utilized a batch size of [Insert your batch size, e.g., 8] and a step-decay learning rate schedule. **Loss Function:** The standard multi-task loss was used, combining binary cross-entropy for contact point classification and L2 loss for grasp pose regression.

Our baseline training graph is displayed in figure 4a (purple curve).

(left) show feasible grasps distributed around the rim and body. The PointNet++ backbone (center) produces candidates that are spatially concentrated on the rim with nearly identical orientations, showing the local nature of its radius-based feature aggregation. The PTv3 backbone (right) distributes candidates across the mug body with more diverse approach directions, consistent with its global attention mechanism capturing a wider range of geometric context. The top-1 selected grasp is highlighted in green (ground truth) and blue (PointNet++, PTv3).

4 Results

4.1 Baseline vs. extension: PointNet++ vs. PTv3

We compare our PTv3 extension (with the `sparse3d` xCPE variant) against the original Contact-GraspNet PointNet++ backbone at the same anchor configuration. Both models share the CGN heads and CGN loss verbatim; only the backbone differs, which isolates the contribution of windowed-attention on serialised voxels (PTv3) over radius-ball grouping with shared MLPs (PointNet++). Final metrics on the held-out *test* split, which contains meshes never seen at training time, are reported in Table 3.

Table 3: Test-split metrics at matched conditions (lower is better).

Metric	PointNet++ (`pn2`)	PTv3 (`sparse3d`)	Δ
`test/loss` (total)	0.82	**0.75**	-0.07
`test/loss_conf`	0.66	**0.62**	-0.04
`test/loss_adds`	0.046	**0.036**	-0.010
`test/loss_width`	**0.42**	0.78	$+0.36$

PTv3 wins on three of the four reported terms, including the headline total loss (a relative reduction of roughly 9%) and the symmetry-aware geometric ADD-S loss on the five gripper keypoints (a relative reduction of roughly 22%). This is consistent with the design hypothesis: voxelisation plus space-filling-curve serialisation gives every attention block access to a genuine 3-D neighbourhood, which is precisely what is needed to localise contact points and recover the gripper pose around them. The drop in `loss_conf` further indicates that the transformer backbone produces better-calibrated per-point graspability scores than the hierarchical sampling-and-grouping baseline.

The single, instructive exception is the gripper-width classifier: PointNet++ achieves a markedly lower `test/loss_width` despite sharing the width head verbatim with PTv3. We interpret this as a failure mode of the transformer backbone on *discrete* predictions: PTv3's windowed attention preferentially aggregates fine-grained local geometry, which helps continuous pose regression (ADD-S) but provides little extra signal for the 10-bin width classification, where PointNet++'s coarser, shared-MLP features apparently generalise more cleanly to novel meshes.

4.1.1 Per-Object Grasp Success Rate

Table 4 reports the top-1 grasp confidence score for five object categories randomly sampled from the held-out test set. For each scene, the model ranks all 4096 point cloud contacts by predicted confidence and we report the score of the highest-ranked contact point.

Table 4: Top-1 grasp confidence score (highest-ranked predicted contact point) on one randomly chosen test scene per object category, evaluated with the PN2 and PTv3 checkpoints.

Object	**CGN (PointNet++)**	**CGT (PTv3)**
Cup	0.5121	**0.7866**
Bottle	0.5049	**0.5090**
Bowl	0.4981	**0.5125**
Mug	0.5049	**0.5161**
Pencil	0.0575	**0.2732**

4.2 Hyperparameter Tuning

To optimize the PTv3/sparse3d configuration, an 80-trial Bayesian hyperparameter sweep was conducted using `val/loss` as the primary objective. The search space and resultant observations are structured below.

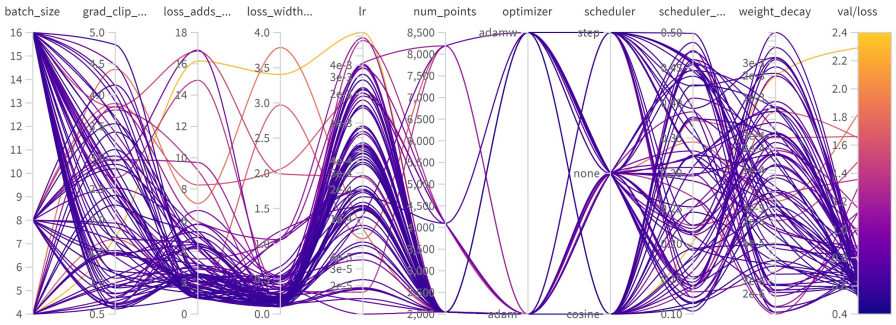


Figure 3: W&B parallel-coordinates plot for the PTv3 hyperparameter sweep (color indicates validation loss).

Table 6: Speed and efficiency comparison of xCPE variations. Lower loss is better.

Variation	Efficiency profile	Latency impact	Loss
`sparse3d`	**Optimal**	Core of the 3.3× speedup	**0.7539**
`knn`	Low	Reintroduces k-NN bottleneck	0.7834
`conv1d`	Highest	Minimal complexity	0.7980

Table 5: Hyperparameter Search Space

Parameter	Search Range / Options
Learning Rate	5×10^{-5} to 3×10^{-3}
Optimizer	Adam, AdamW
Batch Size	4, 8, 16
Scheduler	None, Cosine, Step (with varying decay factors)
Loss Weights	`loss_adds_weight`, `loss_width_weight`
Input Density	`num_points` (2048, 4096, 8192)
Regularization	Weight decay, Gradient-clip max-norm

4.2.1 Key Empirical Observations

The 80-trial sweep identifies specific constraints necessary for model stability and performance:

- **Learning Rate:** Optimal within 2×10^{-4}–10^{-3}. Values exceeding 3×10^{-3} diverge, while those below 5×10^{-5} underfit.
- **Loss Rebalancing:** Correcting CGN's default miscalibration by reducing `loss_adds_weight` to **1–3** and `loss_width_weight` to < 0.2 is critical for convergence.
- **Optimization:** A clear preference for **AdamW** combined with **cosine** or **step** schedules.

4.3 Architecture Variation

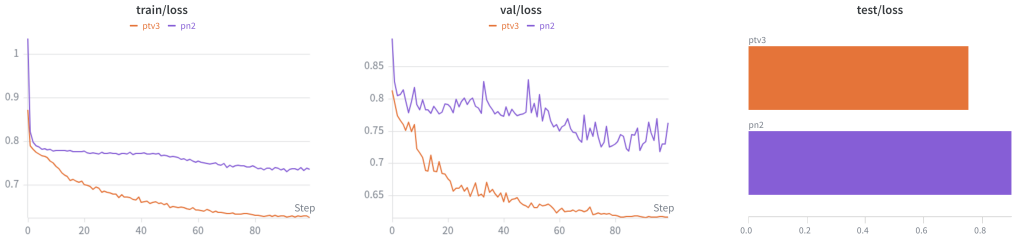
The Point Transformer V3 (PTv3) architecture is designed for computational efficiency, achieving a 3.3× increase in inference speed and a 10.2× reduction in memory consumption compared to Point Transformer V2 (PTv2)[16]. The primary architectural variation explored here is the Extended Conditional Positional Encoding (xCPE) mechanism, which determines how local geometric context is aggregated.

Table 6 summarizes the performance/efficiency trade-offs among three xCPE implementations: `sparse3d`, `knn`, and `conv1d`. 5a shows the training loss curves for all 3 choices.

Key observations.

- **Removing the bottleneck:** `sparse3d` replaces k-Nearest Neighbors (k-NN) with sparse submanifold convolutions, avoiding expensive neighbor searches that limited scaling in PTv2[16] (where k-NN accounted for 28% of the forward-pass time).
- **Latency reduction:** By removing Relative Positional Encoding (RPE) and k-NN dependencies, PTv3 reduces processing time for a standard ScanNet frame from 146 ms to approximately 44 ms.
- **Speed–accuracy trade-off:** `conv1d` maximizes throughput by serializing points into a 1D sequence, but it degrades accuracy because it does not capture spatial geometry as effectively as `sparse3d`.

Overall, `sparse3d` is selected for the final model because it provides the best trade-off: substantially faster than the k-NN approach while remaining markedly more accurate than the `conv1d` shortcut.

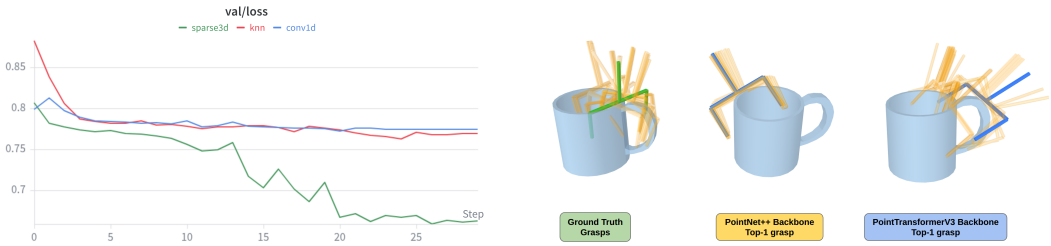


(a) Train loss (b) Validation loss (c) Test loss

Figure 4: Baseline vs Proposed model

Table 7: Quantitative comparison between PointNet++ and PTV3 architectures.

Metric	Baseline (PointNet++)	Proposed (PTV3)
Inference time (ms)	113.64	66.973
Parameter count (M)	0.8	8.8



(a) xCPE variations (b) Grasp predictions on a test mug scene

Figure 5: Grasp predictions on a test mug. Left: Ground-truth grasps on rim and body. Center: PointNet++ produces spatially concentrated rim grasps with uniform orientations due to local feature aggregation. Right: PTv3 distributes diverse grasps via grid-window attention. Top-1 selection: green (ground truth), blue (PTv3). MuJoCo simulation results for predicted grasps can be found at C

4.4 Failure Case Analysis

We observe several recurring failure modes across both models. First, small objects (e.g., pencils) occupy very few voxels after discretization and have relatively few ground-truth grasps, resulting in sparse point clouds and limited supervision. This reduces feature richness and is reflected in lower confidence scores. Second, rotationally symmetric objects admit many geometrically equivalent

grasps. Since the model must commit to a single orientation per contact point, the loss landscape contains multiple near-equivalent optima, which can dilute confidence and distribute probability mass across competing grasp directions. Third, each scene is rendered from a single viewpoint, meaning only one side of the object is observed. As a result, grasps that require access to occluded regions (e.g., a mug handle) are often missed or predicted with low confidence. Finally, some predicted grasps are kinematically infeasible for the Franka Panda robot. For example, horizontal grasps may be valid in the contact-based representation but violate joint limits or cause collisions during execution due to the robot's vertical wrist configuration. Importantly, these failure modes are largely independent of the backbone architecture. The transition to PTv3 introduces no new failure cases; rather, PTv3 resolves several failure cases observed in the PointNet++ baseline while maintaining performance on all previously successful examples.

5 Discussion

Our results show that replacing the PointNet++ backbone with PTv3 improves grasp prediction performance while substantially reducing inference time. We observe a reduction in total loss of over 20% alongside a significant latency improvement, indicating that attention-based feature aggregation can enhance both accuracy and efficiency in 3D grasp prediction.

These improvements are meaningful in the context of robotic systems, where grasp prediction is only one component of a larger perception–action pipeline. More accurate and confident grasp predictions can improve downstream reliability, particularly in the presence of sensor noise and partial observations from depth images or point clouds. At the same time, reduced inference time is critical for real-time deployment, where grasp planning must operate under tight latency constraints.Despite these gains, the model's outputs remain limited by system-level considerations. Predicted grasps must ultimately be executed by a physical robot, requiring feasible trajectories, collision avoidance, and robustness to perturbations. This highlights an important gap between prediction quality and execution feasibility that is not captured by loss-based evaluation alone.

Motivated by the efficiency gains of PTv3, we explored a potential architectural simplification by replacing sparse 3D convolutions with a 1D convolution applied after space-filling curve serialization [5]. This approach reduces explicit spatial aggregation in favor of sequential processing, trading some geometric fidelity for improved computational efficiency. We hypothesize that the imposed ordering from serialization provides sufficient structure for effective feature learning while enabling faster inference. This direction offers a promising path toward further reducing latency and enabling real-time grasping systems.

6 Future Directions

While our current work focuses on improving grasp prediction through architectural changes, a natural next step is to structure representations more intentionally. We draw inspiration from the visual system, where early processing is shared across tasks, and later representations become increasingly specialized for perception (ventral) and action (dorsal). This organization suggests a computational principle: low-level features are broadly useful across tasks, while high-level features become task-specific and may even conflict due to differing invariance requirements.

Motivated by this, we propose a hierarchical feature fusion approach, where embeddings from different layers of a pretrained recognition network (e.g., ResNet) are injected into the grasp model's bottleneck representation, allowing the model to directly incorporate geometric and semantic cues from multiple levels of abstraction when predicting grasps.

This direction is valuable from both engineering and scientific perspectives. For grasping, it offers a practical way to incorporate complementary visual features to improve robustness and generalization. From a neuroscience perspective, it provides a controlled framework to study how representations for perception and action interact, which is difficult to isolate directly in the brain.

7 GitHub Repositories & Project Website

The complete codebase, containing both the baseline and PTv3 implementations, is publicly available at `https://github.com/devanshroyal-7/Contact-GraspTransformer`. Additionally, the repository tracking our midterm progress and initial Contact-GraspNet experiments can be found at `https://github.com/dineshvarunshankar/11785_Project.git`.Grasp visualization outputs, MuJoCo simulation recordings, and additional results are hosted at:`https://sites.google.com/andrew.cmu.edu/contact-grasptransformer/home`.

8 Team Contributions

- **Devansh Royal J.**: PTv3 backbone implementation, data generation, dataloader, inference pipeline, and voxel visualization.
- **Aida Mirebrahimi**: Literature review, project definition, augmentation pipeline, loss formulation, and writing.
- **Tim Wang**: Training loop, checkpointing, learning rate scheduling, evaluation metric, and training.
- **Dinesh Varun Shankar K.**: PointNet++ baseline, Backbones Evaluation, MuJoCo simulation, grasp visualization, and hyperparameter sweeping.

References

[1] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. In *IEEE International Conference on Robotics and Automation*, 2000.

[2] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309, 2014.

[3] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. ACRONYM: A large-scale grasp dataset based on simulation. In *Under Review at ICRA 2021*, 2020.

[4] Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu. Graspnet-1billion: A large-scale benchmark for general object grasping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11444–11453, 2020.

[5] Ben Graham. Sparse 3d convolutional neural networks. *CoRR*, abs/1505.02890, 2015.

[6] Hongzhuo Liang, Xiaojian Ma, Shuang Li, Michael Goerner, Song Tang, Bin Fang, Fuchun Sun, and Jianwei Zhang. Pointnetgpd: Detecting grasp configurations from point sets. In *IEEE International Conference on Robotics and Automation*, 2019.

[7] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In *Robotics: Science and Systems*, 2017.

[8] Douglas Morrison, Peter Corke, and Juergen Leitner. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. In *Robotics: Science and Systems*, 2018.

[9] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *IEEE International Conference on Computer Vision*, 2019.

[10] Domenico Prattichizzo and Jeffrey C. Trinkle. Grasping. In *Springer Handbook of Robotics*, pages 671–700. Springer, 2008.

[11] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, 2017.

[12] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, 2017.

[13] Manolis Savva, Angel X. Chang, and Pat Hanrahan. Semantically-Enriched 3D Models for Common-sense Knowledge. *CVPR 2015 Workshop on Functionality, Physics, Intentionality and Causality*, 2015.

[14] Martin Sundermeyer, Arsalan Mousavian, Rudolph Triebel, and Dieter Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3765–3771, 2021.

[15] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point transformer v3: Simpler, faster, stronger. In *CVPR*, 2024.

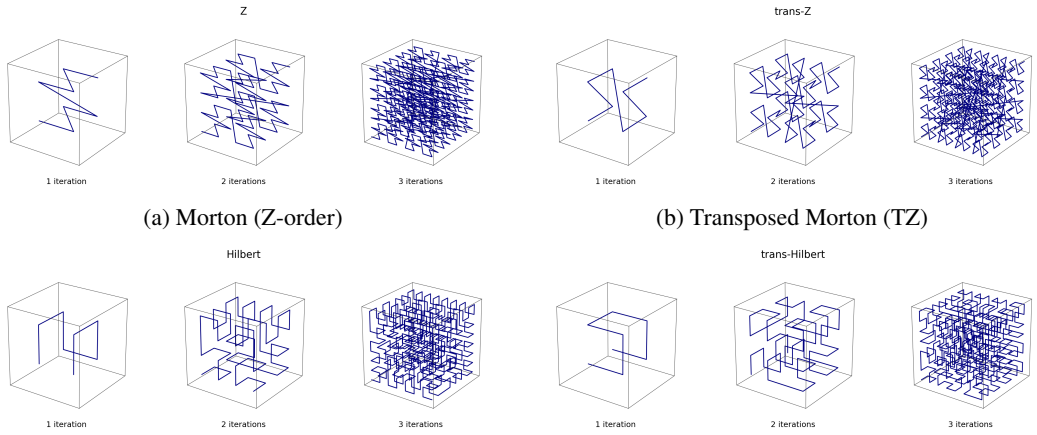
[16] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[17] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip H. S. Torr, and Vladlen Koltun. Point transformer. *CoRR*, abs/2012.09164, 2020.

Appendices

A Space-Filling Curves

Figure 6 shows the four space-filling curves used by PTv3 for multi-pattern serialization.



(a) Morton (Z-order)

(b) Transposed Morton (TZ)

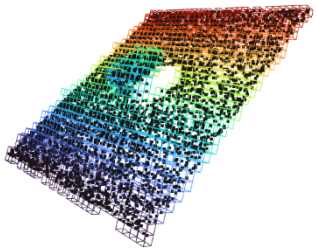
(c) 3-D Hilbert

(d) Transposed 3-D Hilbert

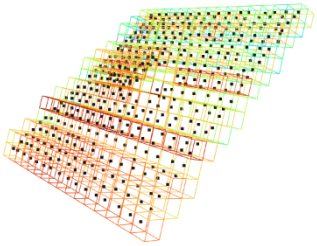
Figure 6: Space-filling curve orderings used for point serialization.

B Encoder Stage Visualizations

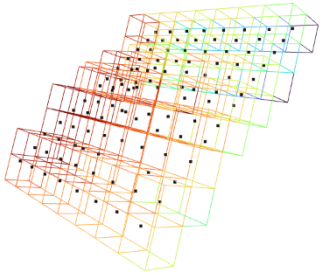
Figure 7 provides a qualitative view of the encoder stages produced by our voxelization pipeline.



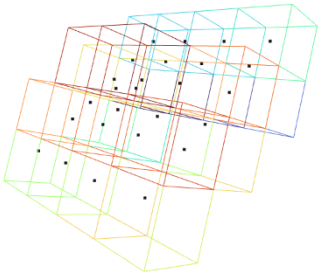
(a) Encoder stage 0



(b) Encoder stage 1



(c) Encoder stage 2

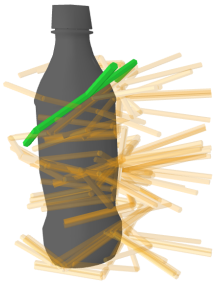


(d) Encoder bottleneck

Figure 7: Encoder images produced by voxelization at successive stages.

C MuJoCo Simulation

To evaluate whether predicted grasps are physically executable, we built a simulation pipeline using MuJoCo. A Franka Emika Panda arm and the target object are placed in a tabletop scene, and the model's top-ranked grasp is converted into a three-phase execution plan: approach the object along the predicted approach vector, close the gripper, and lift vertically by 15 cm. Each phase is reached via iterative inverse kinematics (IK), which solves for a feasible 7-DoF joint configuration given the target end-effector pose. A grasp is successful if the object is lifted by at least 3 cm. Failures are logged with specific reasons (e.g., IK solver failure, object slippage), enabling post-hoc analysis of failure modes not captured by loss-based evaluation.

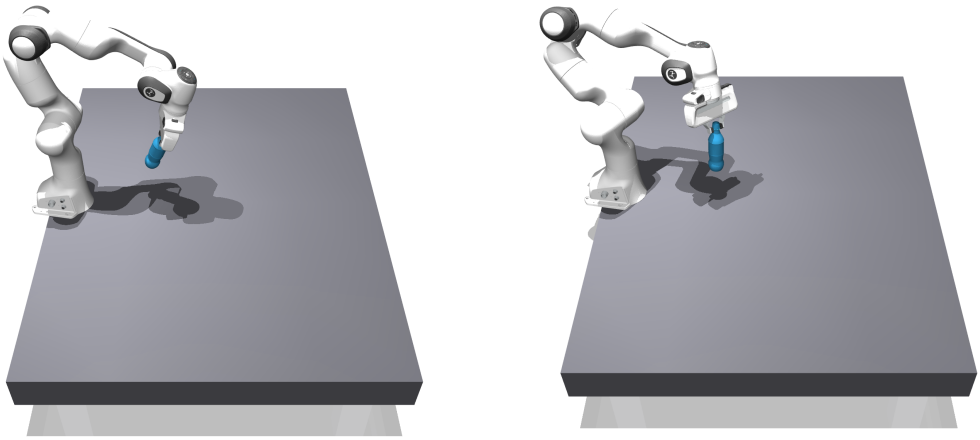


(a) Ground truth

(b) PointNet++

(c) PTv3

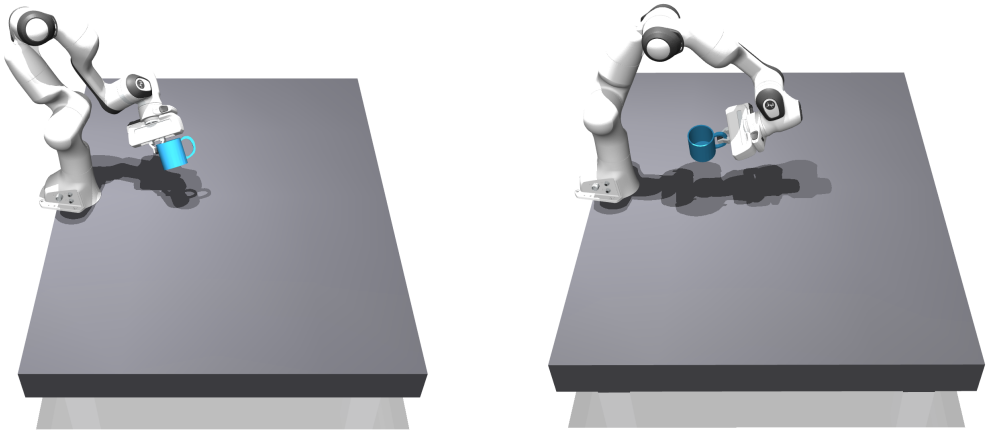
Figure 8: Bottle case study. Comparison of ground-truth grasp and the top-1 predicted grasps from each backbone.



(a) PointNet++ top-1 grasp execution (Bottle)

(b) PTv3 top-1 grasp execution (Bottle)

Figure 9: MuJoCo simulation of the Panda arm executing the top-1 predicted grasp from each backbone on the same test object — Bottle.



(a) PointNet++ top-1 grasp execution

(b) PTv3 top-1 grasp execution

Figure 10: MuJoCo simulation of the Panda arm successfully executing the top-1 predicted grasp from each backbone on the same test object — Mug.