DextER: Language-driven Dexterous Grasp Generation with Embodied Reasoning

Supplementary Material

A. Implementation Details

We provide additional implementation details of DextER.

A.1. Model

Point cloud encoder. We support two pretrained 3D point cloud encoders in our experiments:

- **PartField** [24]: A triplane-based encoder pretrained on Objaverse using contrastive learning with SAM2 masks for part-aware feature extraction. The encoder produces part-geometry aware features beneficial for contact reasoning. We downsample the triplane feature maps at the encoder bottleneck, yielding 768 visual tokens ($3 \times 16 \times 16$).
- **Uni3D** [51]: A transformer-based point cloud encoder pretrained on OpenShape [23] dataset. This encoder provides global object-level features, where we use FPS sampling with 512 points processed through pretrained CLIP vision transformer layers to produce visual tokens. We use the 'base' variant of the encoder.

Both encoders process input point clouds sampled to 10,000 points with XYZ coordinates.

Tokenization. We discretize continuous values into tokens:

- **Action tokens**: Grasp actions are normalized using quantile normalization (1st-99th percentile $\rightarrow [-1, 1]$), then uniformly discretized into $N_{\mathbf{a}} = 256$ bins per dimension.
- **Position tokens**: Contact positions are bounded by dataset-specific min/max values computed from the training set, then uniformly discretized into $N_{\text{pos}} = 256$ bins per dimension.
- **Link tokens**: Shadow Hand link names are encoded as special tokens (e.g., `<rh_ffdistal>`, `<rh_palm>`, `<rh_thdistal>`).
- **Delimiter tokens**: Special tokens mark sequence boundaries: `<|contact_start|>`, `<|contact_end|>`, `<|action_start|>`, `<|action_end|>`.

All special tokens are registered in the pretrained tokenizer, expanding the vocabulary while preserving language understanding capabilities.

Attention mechanism. We employ a hybrid attention pattern where point cloud tokens use bidirectional attention to capture global geometric context, while the other tokens follow causal (autoregressive) attention. This design enables comprehensive 3D understanding while maintaining standard next-token prediction for text and actions. The attention mask pattern is illustrated in Fig. 4.

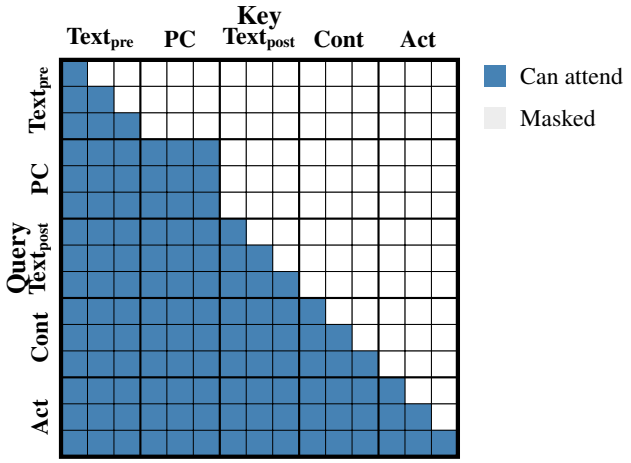

Figure 4. **Prefix-LM attention mask for DextER.** Point cloud (PC) tokens use bidirectional attention (full blue blocks in PC rows/columns), whereas the other tokens use causal attention (lower triangular patterns), attending to all preceding point cloud tokens.

A.2. Training

We initialize the model from pretrained point cloud encoder and LLM backbone checkpoints, then finetune all components end-to-end. The training hyperparameters are summarized in Table 6.

Parameter	Value
Optimization	
Learning rate	1×10^{-4} with cosine decay
Warmup steps	1,000 (linear warmup)
Weight decay	1×10^{-10}
Gradient clipping	Max norm 1.0
Beta parameters	$\beta_1 = 0.9, \beta_2 = 0.95$
Epsilon	1×10^{-8}
Training configuration	
Batch size	64 (8 per GPU $\times$ 8 GPUs)
Training steps	100,000 iterations
Precision	Mixed-precision (bfloat16)
Gradient checkpointing	Enabled
Hardware	8 $\times$ NVIDIA A6000 (48GB)
Framework	PyTorch 2.7+ with HuggingFace

Table 6. Training hyperparameters and configuration for DextER.

Prompt design. We use ChatML-style conversation prompt consistent with the Qwen2.5 model, the base LLM model DextER uses. Table 7 shows an example of the prompt.

Input	<|im_start|>system \n You are Qwen, created by Alibaba Cloud. You are a helpful assistant. <|im_end|> \n <|im_start|>user \n <|vision_start|> <|vision_pad|> <|vision_end|> Think about which hand joints will touch the object and where, then plan the grasp. Query: Hand over the cylinder bottle using all five fingers to grasp the body securely. <|im_end|> \n
Output	<|im_start|>assistant \n <|contact_start|> <|rh_ffdistal|> <|pos_bin_124|> <|pos_bin_112|> <|pos_bin_129|> ··· <|contact_end|> <|action_start|> <|action_bin_171|> <|action_bin_148|> <|action_bin_61|> ··· <|action_end|> <|im_end|> \n

Table 7. **Prompt example for DextER.** Input includes system prompt, vision tokens, and task query. Output contains predicted contacts and action sequences. The model is trained with special delimiter tokens for contact reasoning and actions, where all continuous link positions and actions are normalized as described in Sec. 4.1.


```
{
 "rh_ffdistal": [
  [
   -0.00892519375463392,
   -0.02373632483098695,
   0.05827711843946104
  ]
 ],
 "rh_mfdistal": [
  [
   -0.011531257627117413,
   -0.001366592209635986,
   0.028269318665594304
  ]
 ],
 "rh_lfdistal": [
  [
   -0.019563279557225994,
   -0.019054938198173606,
   -0.019934724694947396
  ]
 ]
}
```


Figure 5. **Contact annotation example.**

A.3. Dataset

Contact annotation. For each grasp in the DexGYS and Dexonomy datasets, we automatically extract contact annotations using MuJoCo physics simulation. We load the Shadow Hand and object models into MuJoCo, execute forward kinematics for each grasp pose, and use precise collision queries between finger links and object meshes to extract the 3D surface positions where each hand link makes contact with the object. Since these annotations are obtained directly from the physics engine, they are physically grounded and accurate to simulator precision. These contact annotations enable our model to reason about grasp contact patterns during training. Fig. 5 shows an example of this annotation.

Grasp instruction annotation. To annotate Dexonomy with grasp instructions, we provide Gemma-3 [31] with multi-view renderings of each hand–object configuration with the annotated contact link set. Conditioned on this input and the structured prompt template in Fig. 11, the model first infers the object's semantic category and identifies the functional part being contacted. It then generates two complementary descriptions: (1) a low-level physical grasp instruction and (2) a high-level functional instruction. For all Dexonomy experiments, we use only the physical grasp instructions. This multi-cue conditioning (visual evidence, contact set, and chain-of-thought prompting) serves as our annotation validity mechanism, as the model must produce descriptions consistent across all input modalities. Fig. 6 presents an example of our grasp instruction annotation.

We note that our primary design choice for Dexonomy annotations is to generate open-vocabulary descriptions that capture fine-grained finger-object contact patterns from visual observations, rather than descriptions limited to a fixed grasp taxonomy. While incorporating taxonomy labels could enhance semantic expressiveness, our image-conditioned VLM approach yields more detailed, contact-aware descriptions that enable precise control in downstream tasks.

A.4. Evaluation

Evaluation metrics. Following [38], we evaluate generated grasps using the following metrics:

- **P-FID**: Fréchet Distance between point cloud features [27] of generated and reference grasps, measuring distributional similarity.
- **Chamfer Distance (CD)**: Average spatial discrepancy between generated and ground-truth hand meshes.
- **Contact Distance (Con.)**: L2 distance between predicted and target contact maps on the object surface.
- **Success Rate**: Percentage of grasps that remain stable in Isaac Gym simulation after execution.
- $\mathbf{Q}_1$: Force-closure quality metric measuring the minimum wrench required to break grasp stability.
- **Penetration (Pen.)**: Maximum penetration depth between hand mesh and object point cloud.
- **Diversity** (δ_t, δ_r, δ_q): Standard deviation of palm translation, rotation, and joint angles across generated samples.

P-FID, CD, and Con. measure how well grasps align with task intentions, while Success Rate, Q_1, and Pen. assess physical plausibility and stability. Diversity metrics evaluate the model's ability to generate varied grasp configurations.

Success rate criteria. We evaluate grasp success using

different criteria depending on the benchmark:

- **DexGraspNet** [38]: A grasp succeeds if the hand maintains stable contact with the object for 100 simulation steps under at least one of six gravity directions, with maximum penetration depth ≤ 0.1 cm.
- **DexGraspBench** [7]: A grasp succeeds if it resists six external forces in MuJoCo simulation without severe penetrations (max 1 cm), and the object's final pose remains within 5 cm translation and 15° rotation from its initial configuration.

Dexonomy data splits. To analyze both taxonomy-level and object-level generalization, we structure the Dexonomy dataset into five splits with the following sample counts:

- **Training**: 182,061
- **Seen Objects & Taxonomy**: 12,929
- **Unseen Objects** : 12,929
- **Unseen Grasp Taxonomy**: 12,929
- **Unseen Both**: 12,929

The definitions of seen and unseen grasp taxonomies and objects are as follows:

- **Seen grasp taxonomies**: Small Diameter, Medium Wrap, Power Disk, Power Sphere, Precision Disk, Precision Sphere, Tip Pinch, Light Tool, Writing Tripod, Parallel Extension, Lateral Tripod, Quadpod, Stick, Prismatic 4 Finger, Prismatic 3 Finger, Tripod, Fixed Hook, Lateral, Index Finger Extension, Extensior Type, Palmar, Ring, Ventral, Inferior Pincer
- **Unseen grasp taxonomies**: Large Diameter, Adducted Thumb, Prismatic 2 Finger, Adduction Grip, Sphere 4 Finger, Sphere 3 Finger
- **Seen objects**: Objects included in the training portion of the Dexonomy dataset.
- **Unseen objects**: Objects included in the test portion of the Dexonomy dataset.

Figure 6. Grasp instruction annotation for the Dexonomy dataset.

Figure 7. **Full vs. partial + noisy point cloud observations.** Left: full point cloud. Right: partial point cloud obtained via HPR from two virtual cameras with added sensor noise, retaining only ∼35% of points.

A.5. Robustness to Partial Observations

To evaluate robustness under realistic sensing conditions, we simulate partial point cloud observations using the Hidden Point Removal (HPR) algorithm [17], which determines point visibility from a given viewpoint. We place two virtual cameras following common VLA hardware setups: (1) an *egocentric* wrist-mounted camera positioned behind the hand looking toward the object, and (2) a *third-person* camera providing an overhead/side view. This dual-camera setup retains only ∼35% of the original points visible from these viewpoints. We further add depth-dependent Gaussian noise (σ_d=4mm, scaling linearly with distance) and lateral noise (σ_l=1mm) to simulate real sensor artifacts.

This evaluation is conducted in a zero-shot setting without retraining on partial or noisy inputs. As shown in Table 8, performance degrades only slightly (Success rate -2%, P-FID $+5$%, CD $+3$%) despite the severe reduction in observed points and added noise, demonstrating that DextER generalizes robustly beyond perfect point cloud observations.

	Full	Partial + Noisy
Success↑ (%)	67.14	65.77
P-FID↓	0.20	0.21
CD↓	1.46	1.50

Table 8. **Robustness to partial and noisy observations on DexGYS.** Zero-shot evaluation without retraining on partial inputs.

B. Qualitative Analysis

We provide additional qualitative analysis of DextER on the DexGYS (Fig. 8) and Dexonomy datasets.(Fig. 9) We further present qualitative results for steerable grasp generation. As illustrated in Fig. 10, increasing the number of ECoT constraints from Steer-1 to Steer-5 progressively guides the model toward producing grasps that more closely resemble the ground-truth configuration.

B.1. Failure Mode Analysis

We identify two consistent failure patterns in DextER:

Penetration-induced failures. Quantization artifacts from discretizing continuous contact and grasp parameters into token bins account for 14.7% of failure cases. In these cases, the predicted grasp is semantically correct—the model correctly identifies which fingers should contact which object regions—but the resulting hand configuration exhibits slight penetration with the object mesh, causing simulation failure. We intentionally avoid geometric post-processing (*e.g.*, optimization-based refinement commonly used in baseline methods) to analyze the pure effect of autoregressive contact reasoning.

Unstable grasps on unseen taxonomy. In unseen grasp taxonomy experiments, DextER predicts plausible poses (reflected in low Chamfer Distance and P-FID), but physical execution becomes unstable (*e.g.*, object shaking during simulation). This suggests that while the model generalizes contact reasoning to novel grasp types, subtle structure-dependent stability cues—such as force closure properties specific to certain grasp taxonomies—are not fully captured by the training supervision.

B.2. Steerable Generation: Practical Use Cases

Our contact-based steering interface offers fine-grained, unambiguous control over grasp synthesis. Given full point cloud observations, sampling contact points on the object surface is straightforward using standard processing (*e.g.*, farthest point sampling, region selection). This interface provides significantly more precise control than implicit text descriptions and is valuable in several practical scenarios: (1) collecting diverse manipulation data by varying contact specifications, (2) task-conditioned grasping (*e.g.*, gripping a tool at a functional position), (3) planning grasps in constrained environments where specific contact regions must be avoided, and (4) debugging grasp failures by systematically varying contact constraints.

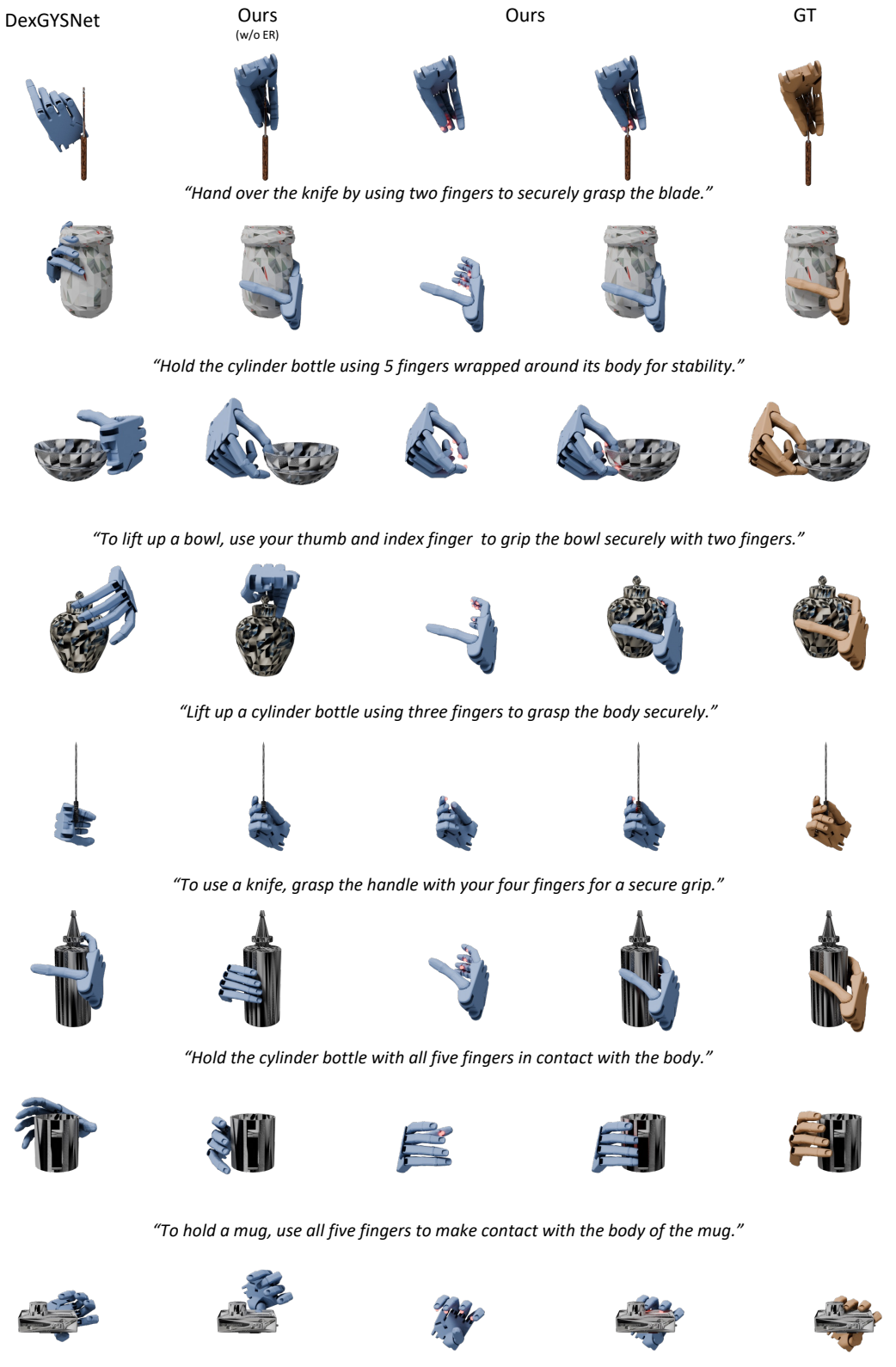

Figure 8. Qualitative results on DexGYS dataset.

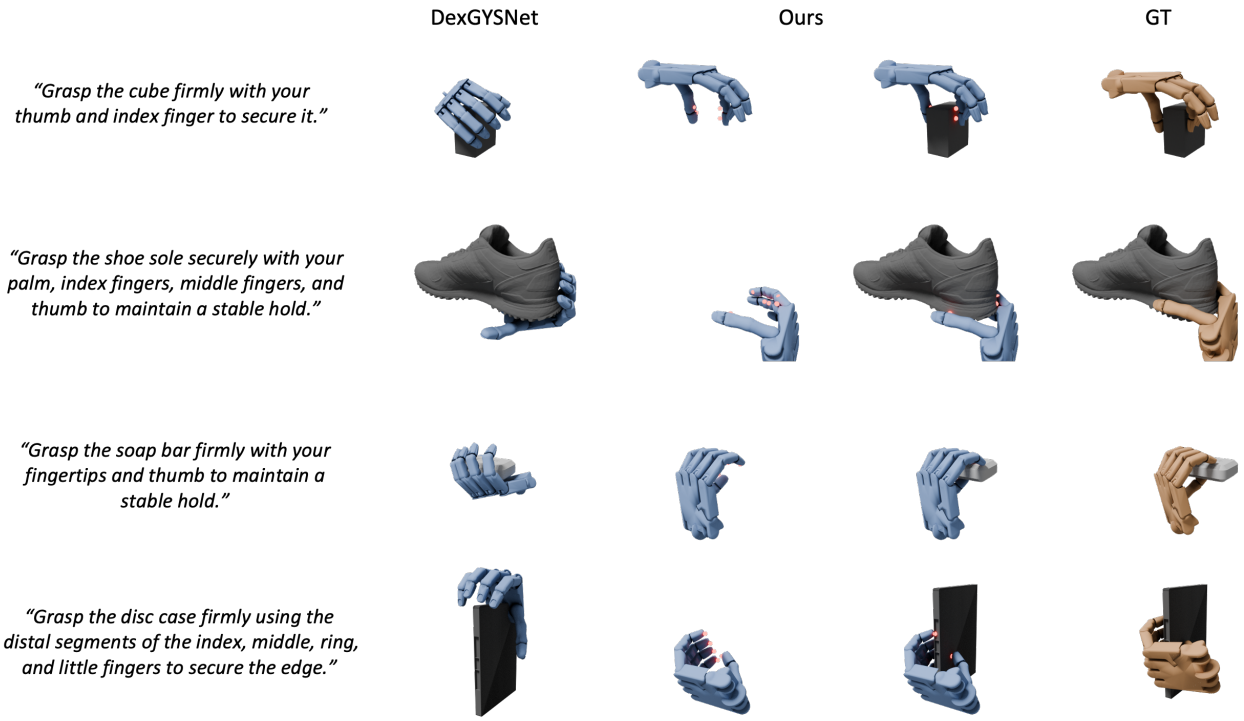

Figure 9. Qualitative results on Dexonomy dataset.

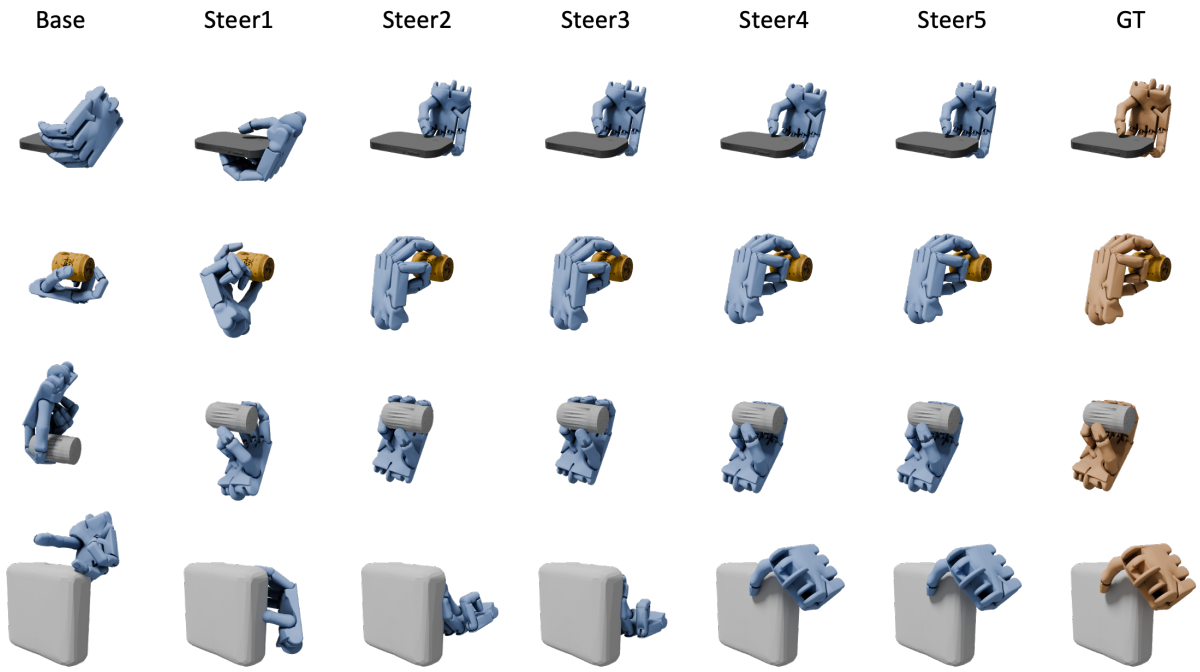

Figure 10. Steerable grasp generation example.

[SYSTEM_PROMPT]
You are a grasp instruction generation system.
You take as input multiple rendered views of an object grasped by a hand, along with:
- The contact link set (which hand links are touching the object)
- Optionally, the object's identifier (If the object class cannot be identified directly (e.g., identifier only contains numbers), infer it.)

Your tasks are:
1. Identify the semantic category of the object. If an object identifier is provided, derive the corresponding object class by parsing it.
2. Identify which functional part(s) of the object are being touched by the hand based on the contact link set.
(e.g., handle, rim, body, lid, trigger, spout, neck, cap, surface, edge, etc.)
3. Generate TWO levels of grasp instructions:
- Level-1: Physical grasp instruction
* Begin with an action verb (e.g., "hold", "lift", "grasp") that matches the observed configuration.
* Clearly describe both the object and the functional part(s) being contacted.
* Must describe how the hand physically interacts with the object.
* Must be ONE sentence
* Examples :
"Lift up the bottle using all four fingers to grasp its body securely."
"To use a trigger sprayer, grasp the bottle body with your thumb and little finger, and pull the trigger with your forefinger."
"Hold the cup by wrapping your fingers around the handle while supporting it with your thumb."
"Hand over a cylinder bottle by grasping it with 4 fingers around the body."
- Level-2: High-level functional instruction
* Begin with an action verb that clearly expresses the intended purpose of the grasp (e.g., throw, pour, carry, hand over, rotate, twist, open, lift, press)
* Do NOT describe which hand link touches which part
* Instead describes the PURPOSE or use-case enabled by this grasp
* Must be ONE sentence
* Examples:
"Carry the bowl steadily across the table."
"Pour liquid smoothly from the cup."
"Hand the tool safely to another person."

You may reference the hand structure below to reason about contact patterns:
Hand Reference:
- PALM: rh_palm
- INDEX FINGER: rh_ffproximal, rh_ffmiddle, rh_ffdistal
- MIDDLE FINGER: rh_mfproximal, rh_mfmiddle, rh_mfdistal
- RING FINGER: rh_rfproximal, rh_rfmiddle, rh_rfdistal
- LITTLE FINGER: rh_lfproximal, rh_lfmiddle, rh_lfdistal
- THUMB: rh_thproximal, rh_thmiddle, rh_thdistal
- PALM: rh_palm
- Proximal: The finger segment closest to the palm.
- Middle: The middle finger segment.
- Distal: The fingertip segment, used for precise contact.

[USER_PROMPT]
These are {n_views} rendered views of an object grasped by a hand.
The hand makes contact with the object through the following links: {contact_links}.
Optional object's identifier : {object_class}

You must:
1) Identify or infer the semantic class of the object.
2) Identify which functional part(s) of the object are being touched by the hand based on the contact link set.
3) Generate {n_queries} LOW-LEVEL physical grasp instructions (Level-1):
- Begin with an action verb (e.g., hold, grip, pinch, grasp, support)
- ONE sentence describing how the hand interacts with the object's functional parts based on grasp type
4) Generate {n_queries} HIGH-LEVEL functional instructions (Level-2):
- Focus on the PURPOSE enabled by this grasp (e.g., lifting, throwing, pouring, handing over)
- Do NOT mention specific hand links, contact points, or finger segments
- ONE sentence per instruction

Output the result strictly in JSON format.

Figure 11. Prompt template for grasp instruction generation in Dexonomy dataset.