

APPENDIX

This appendix provides supplementary materials supporting the main paper, organized as follows:

- **LLM Usage Disclosure:** Role specification of large language models
- **Experimental Setup:** RL Bench tasks, implementation details and baselines
- **Time Analysis:** Additional computational efficiency analysis
- **Keypoints Visualization:** Qualitative results of geometrically consistent keypoints
- **Egocentric Rendering Visualization:** Position-aware pretraining data samples

A LARGE LANGUAGE MODEL USAGE DISCLOSURE

In compliance with ICLR 2026 policy, we disclose the use of large language models (LLMs) in the preparation of this work:

- DeepSeek-R1 (<https://www.deepseek.com>) was utilized exclusively for **language polishing** of non-technical sections (Introduction and Related Work).
- All technical content (Method, Experiment and Conclusion) was written by humans without LLM assistance.
- LLM-generated text was rigorously verified and modified by the authors.
- No LLM was used for data analysis, algorithm design, or scientific interpretation.

The authors assume full responsibility for all content in this manuscript.

B EXPERIMENTAL SETUP

This section specifies the experimental framework covering RL Bench tasks, real-robot setup, our implementation details, baseline architectures and processing pipelines.

B.1 RL BENCH TASKS

We briefly summarize the RL Bench tasks in Table 3, comprising 18 tasks with 249 variations across object color, category, placement, count, shape, and size. Each task requires executing manipulation sequences such as pick-and-place, tool use, drawer opening, and precision operations like peg insertion and shape sorting. During evaluation, the robot handles variations including novel object poses, randomly sampled language instructions, and scenes with unseen object appearances. This task variety necessitates manipulation policies with generalizable comprehension of scenes and instructions, along with adaptable skill acquisition beyond specialized adaptation to individual scenarios.

For a more detailed introduction of each task, please refer to PerAct (Shridhar et al., 2023). For training and evaluating Cortical Policy, we render four virtual camera views to get visual inputs, including 3 static viewpoints and 1 dynamic viewpoint. We visualize the rendered images in Fig. 5.

B.2 REAL-ROBOT EXPERIMENTAL SETUP

To evaluate Cortical Policy in real-world scenarios, we deploy a tabletop manipulation system consisting of a dual-arm Cobot Agilex ALOHA robot. As shown in Fig. 6, the experimental setup integrates two fixed cameras for static-view perception, complemented by two wrist-mounted cameras for dynamic-view perception. In our experiments, we utilize a single arm to execute four distinct manipulation tasks: one benchmark task aligned with RL Bench and three new tasks designed to test spatial reasoning and dynamic scene adaptation abilities. Each task collects 45 human-teleoperated demonstrations with placement variations. Four real-world tasks are detailed as follows:

- **Stack 2 blocks:** This basic task requires the robot to sequentially stack a yellow block onto an orange block, corresponding to RL Bench “stack blocks” task for sim-to-real transfer evaluation.

Table 3: Summary of the 18 RL Bench tasks for multi-task experiments.

Task Name	Language Template	#of Variations	Variation Type
close jar	"close the __ jar"	20	color
drag stick	"use the stick to drag the cube onto the __ target"	20	color
insert peg	"put the ring on the __ spoke"	20	color
meat off grill	"take the __ off the grill"	2	category
open drawer	"open the __ drawer"	3	placement
place cups	"place __ cups on the cup holder"	3	count
place wine	"stack the wine bottle to the __ of the rack"	3	placement
push buttons	"push the __ button, [then the __ button]"	50	color
put in cupboard	"put the __ in the cupboard"	9	category
put in drawer	"put the item in the __ drawer"	3	placement
put in safe	"put the money away in the safe on the __ shelf"	3	placement
screw bulb	"screw in the __ light bulb"	20	color
slide block	"slide the block to __ target"	4	color
sort shape	"put the __ in the shape sorter"	5	shape
stack blocks	"stack __ blocks"	60	color, count
stack cups	"stack the other cups on top of __ the cup"	20	color
sweep to dustpan	"sweep dirt to the __ dustpan"	2	size
turn tap	"turn __ tap"	2	placement

- **Stack 2 blocks in between the bottles:** This task is an extended version of the basic task, testing the understanding of spatial relationships by requiring the robot to: (1) Precisely place the orange block in the region between two bottles. (2) Stably stack the yellow block atop the orange block.
- **Stack 2 blocks with target displacement:** This task introduces real-world unpredictability, evaluating how effectively the dynamic-view stream handles trajectory adaptation. While the robot is approaching the first block, it is displaced, requiring adaptive trajectory re-planning to complete the original stacking task.
- **Stack 2 blocks with base displacement:** This task also tests the dynamic adaptation capability by displacing the already-stacked orange block during the yellow block’s approach phase. The robot must re-locate the orange block and put the yellow block on it.

For each real-world task, 10 independent trials are conducted to calculate the overall success rate. A trial was considered successful only if all sub-actions of the task are executed correctly.

B.3 IMPLEMENTATION DETAILS

All models are trained on 8 NVIDIA A800 GPUs. We measure the training efficiency of Cortical Policy on an NVIDIA A800 GPU, revealing that VGGT-based 3D supervision generation constitutes the most computationally intensive component. As shown in Fig. 4 (a), the average time costs for VGGT feature aggregation, VGGT decoding, and geometrically consistent keypoint extraction are 1.00×10^{-2} , 8.80×10^{-3} , and 8.92×10^{-3} minutes respectively, resulting in a total of 3.09×10^{-2} minutes for the complete 3D supervision generation. This process is $4.7\times$ slower than the action reasoning procedure of Cortical Policy, primarily due to the computational demands of VGGT inference. To mitigate this bottleneck, we implement a multi-stage strategy that decouples 3D supervision generation from feature consistency optimization. Specifically, geometrically consistent keypoints are precomputed from VGGT, stored, and indexed by their corresponding demonstration IDs.

Cortical Policy is trained for 32.5K steps using the 8-bit LAMB optimizer (Dettmers et al., 2022) with a cosine learning rate decay schedule and 2K-step warm-up. We select the final converged model for evaluation. For baseline methods excluding RVT-2, we report evaluation results from their original publications, with the performance of Hiveformer reported by Chen et al. (2023). Given the architectural similarities between our framework and RVT-2, we conduct a controlled comparison by training RVT-2 from scratch under identical conditions as ours, including the same computing resources and matching hyperparameters (32.5K training steps with batch size 512). This implementation differs from the pretrained RVT-2 model (~ 80 K training steps with batch size 192) released by Goyal et al. (2024), ensuring a fair assessment of our methodological contributions. To

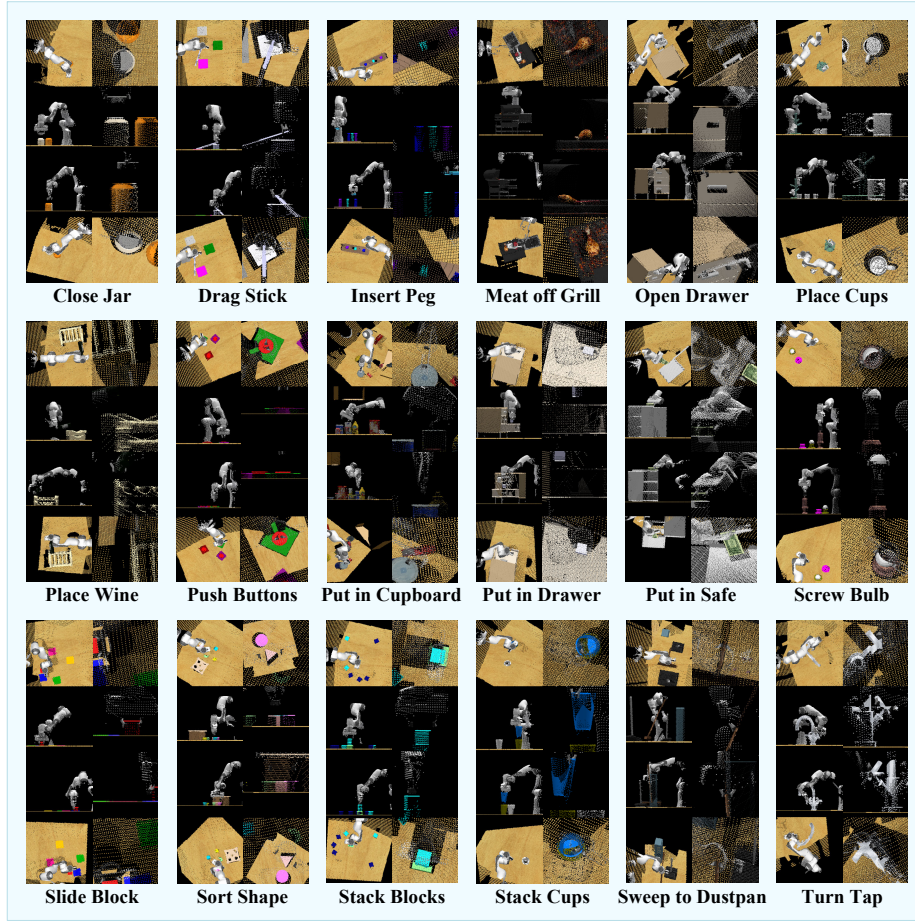


Figure 5: **Rendered views for 18 RL Bench tasks.** Three orthographic static cameras and a dynamic camera (defined by the wrist camera’s raw extrinsic parameters) are used to generate image inputs. For each task, the first three lines show the static views (top, front, right), and the last line shows the dynamic view; rendered results of the first (coarse) stage are presented in the left part while that of the second (fine) stage are shown right.

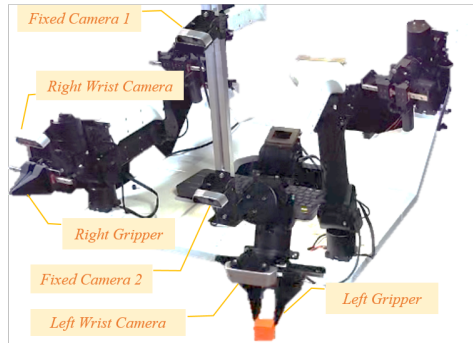


Figure 6: Real-world setup.

account for the randomness in RL Bench’s sampling-based motion planner, we perform three independent test runs per model, each comprising 25 episodes per task. The resulting average success rates with standard deviations are reported in Tables 1 and 2.

Table 4: Training Hyperparameters of Cortical Policy.

Hyperparameters	Value
Batch size	512
Learning rate	5.44×10^{-3}
Optimizer	LAMB
Learning rate schedule	cosine decay
Weight decay	1×10^{-4}
Warm-up steps	2000
Training steps	32.5K
Training epochs	104
\mathcal{L}_{cgc} loss weight (λ)	1
Negative set distance threshold (ζ)	0.1
Keypoints per view (M)	300
Sigmoid temperature (τ)	0.01
Number of static views (N)	3
GLC training epochs	15

We implement data augmentation protocols consistent with established view transformers (Goyal et al., 2023; 2024). For translational augmentation, point clouds are randomly perturbed within ± 12.5 cm along each Cartesian axis. For rotational augmentation, point clouds undergo random z -axis rotations bounded by $\pm 45^\circ$. Table 4 details our training configuration, including a batch size of 512 (64×8) and a learning rate scaling with batch size as $1.0625 \times 10^{-5} \times \text{bs}$.

B.4 BASELINES

This section details the baseline manipulation policies, analyzing their view processing architectures and vision-to-action mapping frameworks.

(1) Hiveformer (Guhur et al., 2022) predicts actions conditioned on a natural language instruction, visual observations at t steps (RGB images, point clouds and proprioception from wrist, left shoulder, and right shoulder cameras) and previous actions at t steps (gripper translation, rotation, and open/close state). Multi-modal tokens are formed by concatenating word tokens and visual tokens from all camera views with embeddings of camera ID, step ID, modality type, and patch location. A transformer encoder then models relationships among camera views, observations and instructions, current and history information. Finally, a CNN decoder predicts rotation and gripper state, while a UNet decoder predicts translation.

(2) RVT (Goyal et al., 2023) re-renders original visual observations (RGB-D images from front, left shoulder, right shoulder, and wrist cameras) into five static virtual viewpoints anchored at the robot base (front, top, left, right, back). This generates 7-channel images: 3 for RGB, 1 for depth, and 3 for pixel coordinates. These re-rendered images, along with language instruction and gripper state, are processed by a joint transformer that sequentially computes intra-view attention, cross-view attention and vision-language attention. The model outputs view-specific heatmaps for predicting 3D translation, and outputs global features that concatenate all viewpoints for estimating gripper rotation, state, and collision indicator.

(3) VIHE (Wang et al., 2024) employs a multi-stage view rendering and action refinement framework comprising an initial global stage and two refinement stages. The initial stage replicates RVT’s five-camera rendering, while the subsequent stages autoregressively generate five virtual in-hand views attached to the previously predicted gripper pose, enabling progressively finer workspace focus. The view transformer adopts masked self-attention to facilitate intra-stage and cross-stage interactions among language instructions, proprioception, multi-stage and multi-camera tokens. During refinement, relative transformations are predicted to update prior stage outputs (gripper poses, collision indicators, and states). Final action predictions are derived from the last refinement stage.

(4) RVT-2 (Goyal et al., 2024) extends RVT with a two-stage architecture: the coarse stage predicts area of interest, while the fine stage renders close-up images for precise gripper pose estimation. Beyond this multi-stage design, RVT-2 improves computational and memory efficiency through replacing transposed convolutions with convex upsampling, optimizing network parameters, sub-

stituting PyTorch3D with a point-renderer for virtual rendering, and utilizing both global and local features to predict gripper rotation, state and collision indicator. Additionally, it reduces five static virtual viewpoints to three (front, top, right), accelerating training while maintaining performance.

(5) Σ -agent (Ma et al., 2024) integrates visual and language encoders, multi-view query transformer (MVQ-Former), contrastive imitation learning module. The visual encoder processes five virtual images with intra-view self-attention. Language instructions are encoded using CLIP and projection layers, generating language tokens for cross-attention computation. MVQ-Former transforms visual tokens into view-specific query tokens for two contrastive learning objectives: a state-language one aligns visual and text tokens in a joint embedding space to learn discriminative representations; a (state, language)-future one concatenates current visual, query and language tokens, then processes them through 4 self-attention layers to derive current-state queries. These queries are contrasted against future-state features, which are extracted by feeding next-state images to the visual encoder and applying average pooling. Both objectives augment the standard imitation learning loss during training to enhance representation learning, but are excluded during inference.

(6) SAM-E (Zhang et al., 2024) incorporates the Segment Anything Model (SAM) as a foundational visual perception module. Based on RVT’s rendering strategy, it processes RGB channels through a LoRA-tuned SAM encoder, enabling generation of prompt-guided, object-oriented image embeddings. Concurrently, spatial features are extracted from depth and pixel coordinate channels via a Conv2D layer. These features are channel-wise concatenated with SAM embeddings to form composite view tokens. Combined with language tokens, these view tokens are processed by a transformer through view-wise and cross-view attention mechanisms. This generates enriched visual tokens for action-sequence prediction. Unlike step-by-step paradigms, SAM-E models coherent action sequences by enforcing temporal smoothness in end-effector poses. For translation prediction, it extends view-specific heatmaps with temporal channels. While rotation, state, and collision indicators are derived from view-fused global features following RVT.

(7) 3D-MVP (Qian et al., 2025) aims to augment visual encoder for learning generalizable representations, decomposing the view transformer into an input renderer, an encoder mapping static virtual images to latent embeddings, and an action decoder. Rather than training the RVT architecture from scratch, 3D-MVP adopts a two-stage training paradigm: first pretrains RVT encoder using masked autoencoding on large-scale 3D scene datasets, then fine-tunes it on downstream manipulation demonstrations. The finetuning procedure is identical to RVT training, while the pretraining introduces a MAE decoder to reconstruct all five virtual images from masked multi-camera tokens. This multi-view pretraining scheme produces 3D-aware features robust to occlusions and viewpoint changes, enhancing manipulation performance and robustness to environmental variations.

C DETAILED TIME ANALYSIS

To evaluate computational efficiency, we execute the model on an NVIDIA A800 GPU for 20 trials with a fixed batch size of 512, recording the processing time for each module in Cortical Policy. As shown in Fig. 4 (a), the static-view stream (1.37×10^{-3} minutes) and dynamic-view stream (2.48×10^{-3} minutes) exhibit comparable latency, with the latter consuming approximately $1.8 \times$ more time. Among submodules, the attention computation of dynamic-view stream is more efficient than that of static-view stream, while its heatmap decoding incurs higher latency. This disparity stems primarily from the multi-scale down-sampling in MViT (Fan et al., 2021), which serves as the backbone of GLC model. Although this multi-scale design increases computation, it significantly enhances view heatmap prediction accuracy, as verified by Li et al. (2018). Notably, both streams process inputs faster than the RVT-2 action head (4.06×10^{-3} minutes). Combined with the 3.5% success rate gain over RVT-2 (Table 1), these metrics demonstrate that our dual-stream transformer achieves superior manipulation performance without compromising adaptive responsiveness.

D VISUALIZATION OF GEOMETRICALLY CONSISTENT KEYPOINTS

Fig. 8 shows additional qualitative results of 3D supervision generation, demonstrating the viewpoint-consistent keypoint distributions across eight manipulation tasks. The figure organizes multi-perspective keypoint visualizations as follows:

- *Rows* (top to bottom): Close Jar, Place Cups, Sweep to Dustpan, Insert Peg, Push Buttons, Drag Stick, Screw Bulb, and Stack Blocks
- *Columns* (left to right): Coarse stage (top, front, right views) followed by fine stage (top, front, right views)

E VISUALIZATION OF EGOCENTRIC RENDERING

Fig. 7 compares dynamic-view options for position-aware pretraining data: raw wrist camera views versus rendered views from dynamic virtual cameras. As can be seen, end-effector positions in raw wrist camera views are fixed at constant pixel coordinates. By contrast, rendered views exhibit positional variations, with shadows indicating regions occluded from physical wrist-mounted cameras. Apart from image samples, egocentric video examples are also included in supplementary material.

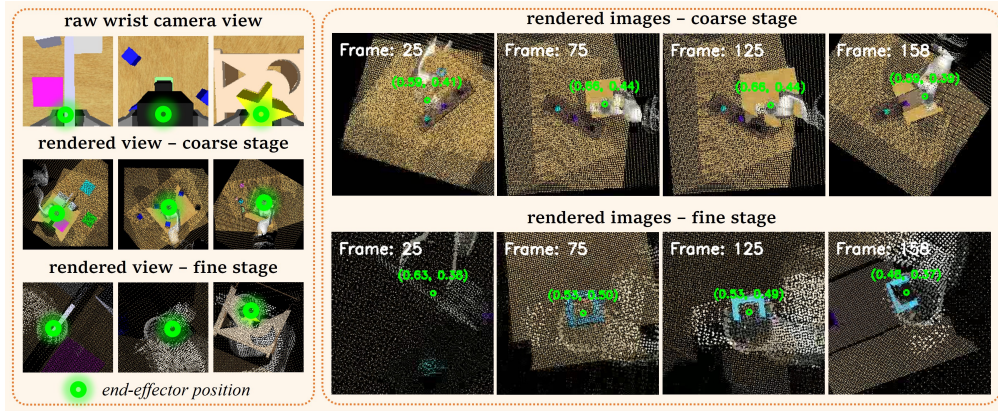


Figure 7: Comparison of dynamic egocentric views and rendered examples.

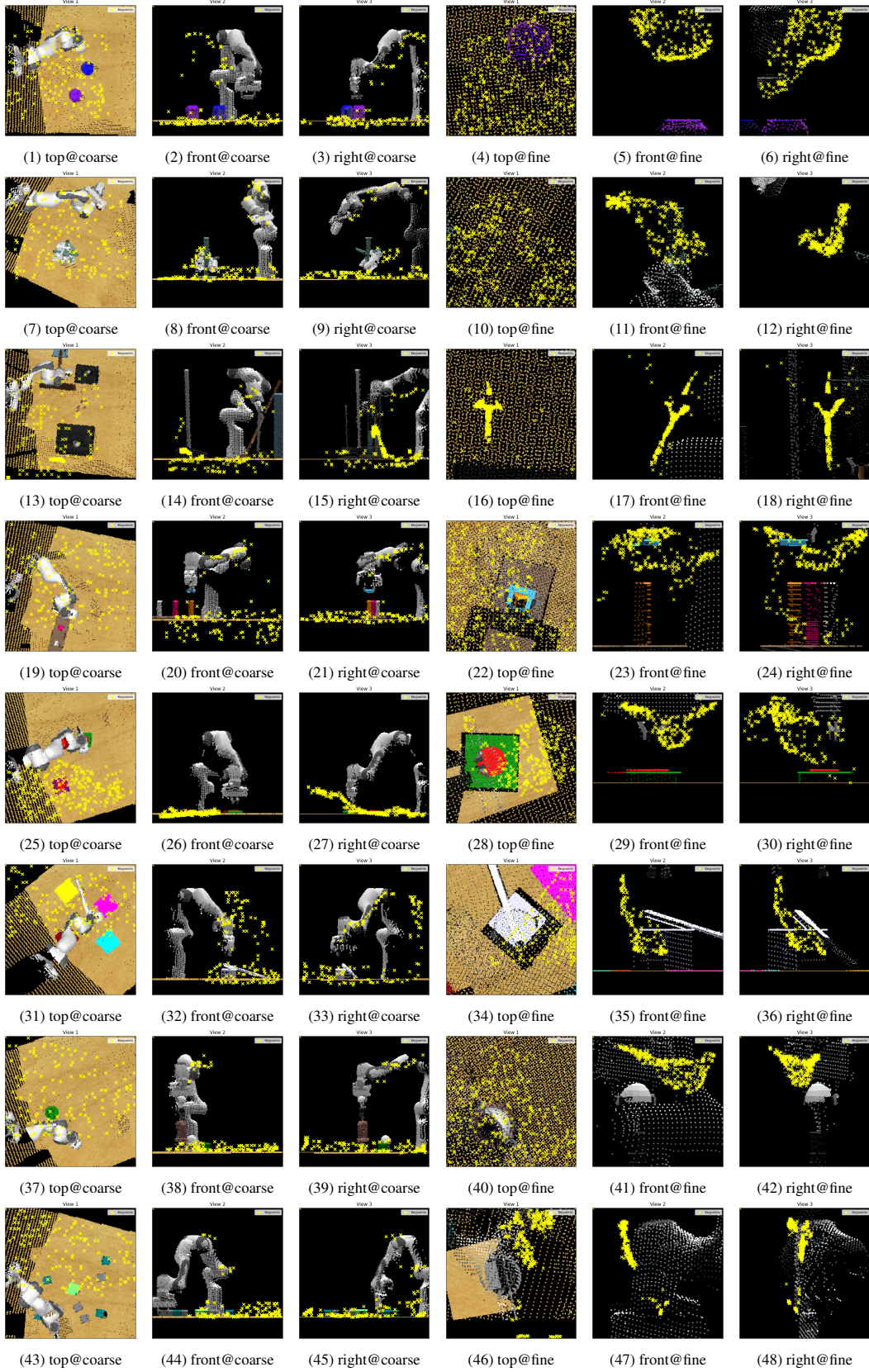


Figure 8: Additional visualization of geometrically consistent keypoints.