

414 **A Task Descriptions**

415 **Simulated tasks.** We select 10 language-conditioned tasks from RLBench [14], all of which involve
 416 at least 2 variations. See Table 5 for an overview. Our task variations include randomly sampled
 417 colors, sizes, counts, placements, and categories of objects, totaling 166 different variations. The set
 418 of colors have 20 instances: red, maroon, lime, green, blue, navy, yellow, cyan, magenta, silver, gray,
 419 orange, olive, purple, teal, azure, violet, rose, black, and white. The set of sizes includes 2 types:
 420 short and tall. The set of counts has 3 instances: 1, 2, 3. The placements and object categories
 421 are specific to each task. For example, `open drawer` has 3 placement locations: top, middle and
 422 bottom. In addition to these semantic variations, objects are placed on the tabletop at random poses
 within a limited range.

Table 5: Language-conditioned tasks in RLBench [14].

Task	Variation Type	# of Variations	Avg. Keyframes	Language Template
<code>close jar</code>	color	20	6.0	"close the — jar"
<code>open drawer</code>	placement	3	3.0	"open the — drawer"
<code>sweep to dustpan</code>	size	2	4.6	"sweep dirt to the — dustpan"
<code>meat off grill</code>	category	2	5.0	"take the — off the grill"
<code>turn tap</code>	placement	2	2.0	"turn — tap"
<code>slide block</code>	color	4	4.7	"slide the block to — target"
<code>put in drawer</code>	placement	3	12.0	"put the item in the — drawer"
<code>drag stick</code>	color	20	6.0	"use the stick to drag the cube onto the — — target"
<code>push buttons</code>	color	50	3.8	"push the — button, [then the — button]"
<code>stack blocks</code>	color, count	60	14.6	"stack — — blocks"

423
 424 **Generalization tasks in simulation.** We design 6 additional tasks where the scene is changed based
 425 on the original training environment, to test the generalization ability of GNFactor. Table 6 gives an
 overview of these tasks. Videos are also available on gnfactor-robot.github.io.

Table 6: Generalization tasks based on RLBench.

Task	Base	Change
<code>drag (D)</code>	<code>drag stick</code>	add two colorful buttons on the table
<code>slide (L)</code>	<code>slide block</code>	change the block size to a larger one
<code>slide (S)</code>	<code>slide block</code>	change the block size to a smaller one
<code>open (n)</code>	<code>open drawer</code>	change the position of the drawer
<code>turn (N)</code>	<code>turn tap</code>	change the position of the tap
<code>push (D)</code>	<code>push buttons</code>	add two colorful jar on the table

426
 427 **Real robot tasks.** In the experiments, we perform three tasks along with three additional tasks where
 428 distracting objects are present. The *oven* task requires the agent to open the door on an oven, a task
 429 which poses challenges due to the precise coordination required. The *faucet* task requires the agent
 430 to rotate the faucet back to center position, which involves intricate motor control. Lastly, the *teapot*
 431 task requires the agent to locate the randomly placed teapot in the kitchen and move it on top of the
 432 stove with the correct pose. Among the three, the teapot task is considered the most challenging due
 433 to the random placement and the need for accurate location and rotation of the gripper. All 6 tasks
 434 are set up in two different kitchens, as visualized in Figure 6. The keyframes used in real robot tasks
 435 are given in Figure 7.

436 **B Implementation Details**

437 **Voxel encoder.** We use a lightweight 3D UNet (only 0.3M parameters) to encode the input voxel
 438 $100^3 \times 10$ (RGB features, coordinates, indices, and occupancy) into our deep 3D volumetric rep-
 439 resentation of size $100^3 \times 128$. Due to the cluttered output from directly printing the network, we



(a) Kitchen 1.

(b) Kitchen 2.

Figure 6: **Kitchens.** We give a closer view of our two kitchens for real robot experiments. The figures are captured in almost the same position to display the size difference between the two.

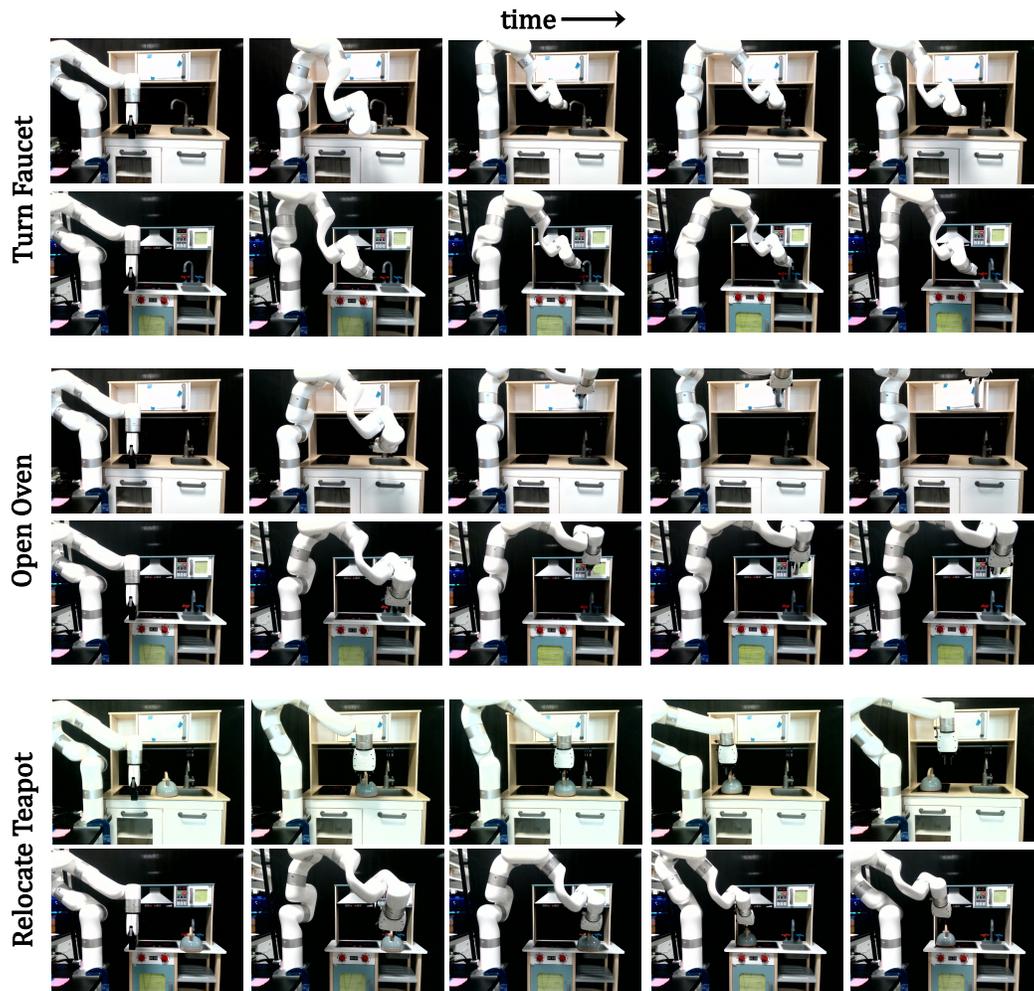


Figure 7: **Keyframes for real robot tasks.** We give the keyframes used in our 3 real robot tasks across 2 kitchens.

440 provide the PyTorch-Style pseudo-code for the forward process as follows. For each block, we use
441 a cascading of one Convolutional Layer, one BatchNorm Layer, and one LeakyReLU layer, which
442 is common practice in the vision community.

```
443 def forward(self, x):  
444     conv0 = self.conv0(x) # 100^3x8  
445     conv2 = self.conv2(self.conv1(conv0)) # 50^3x16  
446     conv4 = self.conv4(self.conv3(conv2)) # 25^3x32  
447  
448     x = self.conv6(self.conv5(conv4)) # 13^3x64  
449     x = conv4 + self.conv7(x) # 25^3x32  
450     x = conv2 + self.conv9(x) # 50^3x16  
451     x = self.conv_out(conv0 + self.conv11(x)) # 100^3x128  
452     return x
```

453 **Neural Radiance Field.** The overall network architecture of our GNF is close to the original
454 NeRF [30] implementation. It mainly consists of 5 ResnetFCBlocks, in which a skip connec-
455 tion is used. The input feature is first projected to 512 with a linear layer and fed into these blocks,
456 and then projected to the output dimension 516 (RGB, density, and Diffusion feature) with a cas-
457 cading of one ReLU function and one linear layer. We provide the PyTorch-Style pseudo-code for
458 the networks as follows.

```
459 GNF(  
460     Linear(in_features=170, out_features=512, bias=True),  
461     (0-4): 5 x ResnetFCBlocks(  
462         (fc_0): Linear(in_features=512, out_features=512, bias=True)  
463         (fc_1): Linear(in_features=512, out_features=512, bias=True)  
464         (activation): ReLU()  
465     ),  
466     ReLU(),  
467     Linear(in_features=512, out_features=516, bias=True)  
468 )
```

469 **Perceiver Transformer.** Our usage of Perceiver Transformer is close to PerAct [3]. We use 6
470 attention blocks to process the sequence from multi-modalities (3D volume, language token, and
471 robot proprioception) and output a sequence also. The usage of Perceiver Transformer enables
472 us to process the long sequence with computational efficiency, by only utilizing a small set of
473 latents to attend the input. The output sequence is then reshaped back to a voxel to predict the
474 robot action. The Q-function for translation is predicted by a 3D convolutional layer, and for the
475 prediction of openness, collision avoidance, and rotation, we use global max pooling and spatial
476 softmax operation to aggregate 3D volume features and project the resulting feature to the output
477 dimension with a multi-layer perception. We could clarify that the design for the policy module is
478 not our contribution; for more details please refer to PerAct [3] and its official implementation on
479 <https://github.com/peract/peract>.

480 C Demonstration Collection for Real Robot Tasks

481 For the collection of real robot demonstrations, we utilize the HTC VIVE controller and basestation
482 to track the 6-DOF poses of human hand movements. We then use triad-openvr package¹ to employ
483 SteamVR and accurately map human operations onto the xArm robot, enabling it to interact with
484 objects in the real kitchen. We record the real-time pose of xArm and 640 × 480 RGB-D observations
485 with the pyrealsense2². Though the image size is different from our simulation setup, we use the
486 same shape of the input voxel, thus ensuring the same algorithm is used across the simulation and
487 the real world. The downscaled images (80 × 60) are used for neural rendering.

¹https://github.com/TriadSemi/triad_openvr

²<https://pypi.org/project/pyrealsense2/>

488 **D Detailed Data**

489 Besides reporting the final success rates in our main paper, we give the success rates for the best single
 490 checkpoint (*i.e.*, evaluating all saved checkpoints and selecting the one with the highest success
 491 rates), as shown in Table 7. Under this setting GNFactor outperforms PerAct with a larger margin.
 492 However, we do not use the best checkpoint in the main results for fairness.

493 We also give the detailed number of success in Table 8 for reference in addition to the success rates
 494 computed in Table 2.

Table 7: **Multi-task test results on RLbench.** We report the success rates for the best single checkpoint for reference. We could observe GNFactor surpasses PerAct by a large margin.

Method / Task	close jar	open drawer	sweep to dustpan	meat off grill	turn tap	Average
PerAct	22.7±5.0	62.7±13.2	0.0±0.0	46.7±14.7	36.0±9.8	
GNFactor	40.0±5.7	77.3±7.5	40.0±11.8	66.7±8.2	45.3±3.8	
Method / Task	slide block	put in drawer	drag stick	push buttons	stack blocks	
PerAct	22.7±6.8	9.3±5.0	12.0±6.5	18.7±6.8	5.3±1.9	23.6
GNFactor	18.7±10.5	10.7±12.4	73.3±13.6	20.0±3.3	8.0±0.0	40.0

Table 8: **Detailed data for generalization to novel tasks.** We evaluate 20 episodes, each across 3 seeds, for the final checkpoint and report the number of successful trajectories here.

Generalization	PerAct	GNFactor w/o. Diffusion	GNFactor
drag (D)	2, 0, 2	15, 2, 5	18, 5, 5
slide (L)	6, 6, 8	1, 10, 10	6, 5, 4
slide (S)	0, 2, 1	6, 1, 5	0, 3, 1
push (D)	6, 3, 3	4, 4, 5	7, 6, 6
open (N)	6, 2, 7	5, 2, 9	8, 5, 6
turn (N)	4, 5, 2	2, 7, 2	6, 6, 5

495 **E Hyperparameters**

496 We give the hyperparameters used in GNFactor as shown in Table 9. **We are committed to re-**
 497 **leasing the code for further details.** For the GNF training, we use a ray batch size $b_{\text{ray}} = 512$,
 498 corresponding to 512 pixels to reconstruct, and use $\lambda_{\text{feat}} = 0.01$ and $\lambda_{\text{recon}} = 0.01$ to maintain ma-
 499 jor focus on the action prediction. We uniformly sample 64 points along the ray for the “coarse”
 500 network and sample 32 points with depth-guided sampling and 32 points with uniform sampling for
 501 the “fine” network.

Table 9: **Hyperparameters** used in GNFactor.

Variable Name	Value
training iteration	100k
image size	$128 \times 128 \times 3$
input voxel size	$100 \times 100 \times 100$
batch size	2
optimizer	LAMB [50]
learning rate	0.0005
ray batch size b_{ray}	512
weight for reconstruction loss λ_{recon}	0.01
weight for embedding loss λ_{feat}	0.01
number of transformer blocks	6
number of sampled points for GNF	64
number of latents in Perceiver Transformer	2048
dimension of Stable Diffusion features	512
dimension of CLIP language features	512
hidden dimension of NeRF blocks	512