

476 A Additional Implementation Details

477 A.1 Model Structure

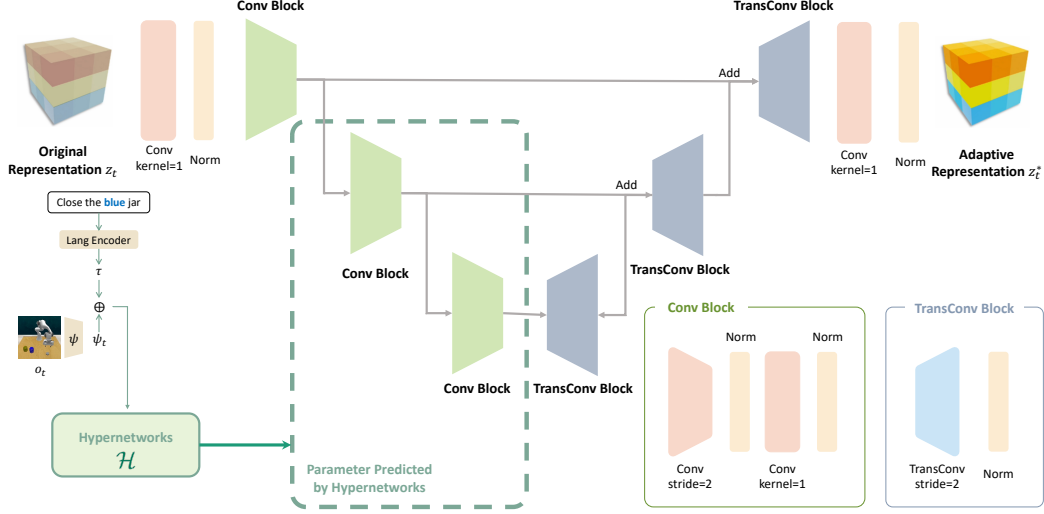


Figure 4: **The detailed model structure of our HyperTASR.** The main diagram shows that the pipeline consists of 2 Convolutional layer and a UNet with skip-connection. *Bottom-right:* The detailed structure of **Conv Block** and **TransConv Block** in the pipeline. These blocks serve as the key components for the encoding and decoding process.

478 HyperTASR we design mainly consists of three convolutional blocks and three transposed convo-
 479 lutional blocks, as detailed in Fig. 4. Each convolutional block contains two convolutional layers,
 480 followed by an InstanceNorm layer and a Leaky ReLU activation function. The first convolutional
 481 layer in each block has a kernel size of 3 and a stride of 2, which reduces the resolution of the feature
 482 map while increasing the feature channel dimension, effectively encoding the features. The second
 483 convolutional layer has a kernel size of 1 and does not change the resolution or channel dimension
 484 of the feature map, serving to refine the encoded features.

485 The transposed convolutional blocks are relatively simpler, consisting of a single transposed convo-
 486 lutional layer followed by InstanceNorm and Leaky ReLU activation. The transposed convolutional
 487 layer increases the resolution of the feature map while reducing the channel dimension, effectively
 488 decoding the features. This layer has a kernel size of 3 and a stride of 2, ensuring that the spatial
 489 dimensions of the feature map are expanded appropriately.

490 HyperTASR used in GNFactor [17] and 3D Diffuser Actor [18] follow the aforementioned structure.
 491 GNFactor directly utilizes a 3D deep volume as its representation, requiring 3D convolutions and
 492 3D transposed convolutions. For the 3D Diffuser Actor, the representation is a point cloud feature,
 493 which in a single-view setup combines a 2D feature map and a depth map, leading us to employ 2D
 494 convolutions and 2D transposed convolutions as the core elements of the UNet architecture.

495 In Fig. 5, we detail the implementation of our hypernetworks. Following [50], we adopt an
 496 optimization-based hypernetwork, which iteratively predicts the parameter updates rather than di-
 497 rectly predicting the final parameter. In our implementation, $K = 8$ represents the parameter update
 498 iteration. To control the parameter size of the UNet and effectively manage the parameter size of
 499 the Hypernetworks, we introduce optional encoders and decoders. The encoder reduces the di-
 500 mensionality of the input features before they are fed into the UNet, while the decoder restores the
 501 dimensionality after processing. This mechanism is particularly useful for maintaining a balance be-
 502 tween model complexity and performance. Specifically, for the 3D Diffuser Actor, we incorporate
 503 these optional encoders and decoders to better adapt to the varying input feature requirements.

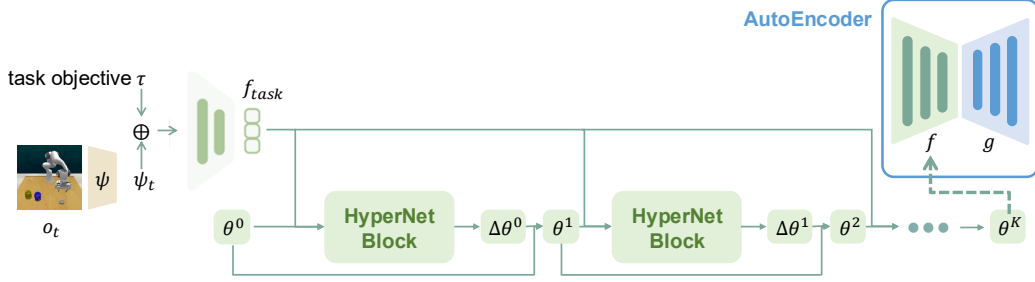


Figure 5: **The detailed structure of hypernetworks we used in HyperTASR.** We employ an optimization-biased hypernetwork that predicts parameter updates iteratively rather than directly generating encoder weights via fully connected layers.

504 A.2 Dataset Composition

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

Table 4: **Dataset composition of 10 manipulation tasks in RLBench [19].**

Task	Variation Type	Language Description Example
place dish	color	“place the — dish on the — tablecloth”
clean cups	color, placement	“Put the — cup into the — basket”
stack cups	color, placement	“stack the — cup on the — cup”
stack blocks	color, count	“stack the — cubes”
put cups on shelf	placement	“put the — cup on the shelf next to — cup”
place blocks	color	“Place the — block on the — plate”

Table 5: **Dataset composition of 6 manipulation tasks in real robot experiments.**

505 We conduct experiments on 10 language-conditioned manipulation tasks from RLBench [19], which
 506 align with the experimental setup of GNFactor [17]. The task variations include randomly sampled
 507 attributes such as colors, sizes, counts, placements, and object categories. Detailed descriptions of
 508 the variation types, variation numbers, average keyframes, and sample language descriptions for
 509 these tasks are provided in Tab. 4.

510 For real robot experiments, we design 6 tasks that cover diverse tasks for “pick and place”. We give
 511 our sample task description in Tab. 5.

512 A.3 Hyperparameters

513 We provide detailed hyperparameters for our experiments in Tab. 6 and Tab. 7, with some parallel
 514 settings and input data differences compared to GNFactor [17] and 3D Diffuser Actor [18]. To
 515 ensure a fair comparison, we reproduce the experiments using the same hyperparameters as the
 516 original codebase and report the corresponding results in Tab. 1. These results serve as a benchmark
 517 for understanding the impact of our modifications.

518 **Impact of Hyperparameter Changes on Experimental Results.** For the GNFactor framework,
 519 we opt not to use distributed data parallel (DDP) training. Instead, we utilize a single GPU, halve

Variable Name	Value
training iteration	200k
image size	$128 \times 128 \times 3$
batch size	1
optimizer	LAMB
learning rate	0.0005
input voxel size	$100 \times 100 \times 100$
number of transformer blocks	6
number of latents in PerceiverI/O	2048
dimension of CLIP language features	512

Table 6: **Hyperparameters** in GNFactor [17] Framework.

Variable Name	Value
training iteration	800k
image size	$256 \times 256 \times 3$
batch size	240
optimizer	Adam
learning rate	0.0001
embedding dim	120
diffusion timestep	100
loss weight of position and rotaion	30 : 20
maximal # of keyposes	25

Table 7: **Hyperparameters** in 3D Diffuser Actor [18] Framework.

the batch size, and double the number of training steps. Despite this adjustment, the final reproduced results fall within the range of multiple experimental outcomes reported in [17]. For the 3D Diffuser Actor, we train both our modified pipeline and the original codebase with the training data provided in the author’s released repository, using an RGB image resolution of 256×256 . Due to the lower resolution compared to the original paper (256×256) [18], our reproduced results (77.0%) are slightly below the original results (78.4%). Additionally, slight adjustments to the loss weights are made to account for the resolution difference, and the best configuration is chosen as the unified hyperparameter setting for all our experiments.

A.4 Computation Cost

We calculate the computation cost of our experiments on GNFactor [17] by measuring both the total number of parameters in the network and the training time, as shown in Tab. 8.

	Model Params	Training Time for 1k steps (s)
GNFactor	64.66M	976.5
Adapter (ours)	99.12M	844.8

Table 8: **Computation Cost.**

For training time, we use an unloaded GPU to train for 1k steps and record the time taken. From the results, we observe that while our network has more parameters, it achieves higher training efficiency. The increased parameter count results from the inclusion of Hypernetworks, but this does not negatively impact training efficiency. On the contrary, by removing the neural renderer used for feature distillation, the overall training time is reduced. This demonstrates that HyperTASR does not impose a significant computational burden on the network and, in some cases, even improves efficiency by eliminating certain supervisory components.

B Additional Results and Analysis

B.1 Additional Ablations Results

In Tab. 3, we only present the success rate data for the ablation studies on the GNFactor [17] framework. Here, we further provide detailed results of these ablation studies across all 10 tasks in Tab. 9. Additionally, we conduct ablation experiments on the 3D Diffuser Actor, and the corresponding results are shown in Tab. 10. These results can validate the effectiveness of our design of HyperTASR.

	Avg. Success	close jar	open drawer	sweep to dustpan	meat off grill	turn tap	slide block	put in drawer	drag stick	push buttons	stack blocks
GNFactor	33.3	32.8	36.0	48.0	51.2	56.8	20.0	8.8	69.6	5.6	4.0
HyperTASR w/ Feature Distillation	34.0	29.6	69.6	35.2	50.4	44.0	8.0	1.6	48.6	43.2	8.8
HyperTASR conditioned on τ	32.2	10.4	47.2	29.6	34.4	51.2	16.8	10.4	92.0	22.4	8.0
HyperTASR predicting θ and ω	36.3	28.0	67.2	20.0	60.8	49.6	20.0	18.4	78.4	7.2	13.6
GNFactor w/ HyperTASR	42.6	32.0	75.2	66.4	48.8	54.4	23.2	22.4	83.2	17.6	3.2

Table 9: **Detailed Ablation Study Results in GNFactor framework.** We report the average success rate across 5 evaluation seeds.

	Avg. Success	close jar	open drawer	sweep to dustpan	meat off grill	turn tap	slide block	put in drawer	drag stick	push buttons	stack blocks
3DDA	79.0	63.2	88.8	94.4	84.8	72.8	94.4	88.8	98.4	87.2	17.4
HyperTASR conditioned on τ	75.4	60.8	75.4	88.8	82.4	59.2	83.2	80.8	89.6	83.2	4.6
HyperTASR predicting θ and ω	79.2	66.4	86.4	96.8	81.4	79.8	84.0	87.2	99.2	89.6	10.4
3DDA w/ HyperTASR	81.3	68.0	87.2	98.4	82.4	85.6	98.4	89.6	100.0	92.0	11.2

Table 10: **Detailed Ablation Study Results in 3D Diffuser Actor (3DDA) framework.** We report the average success rate across 5 evaluation seeds.

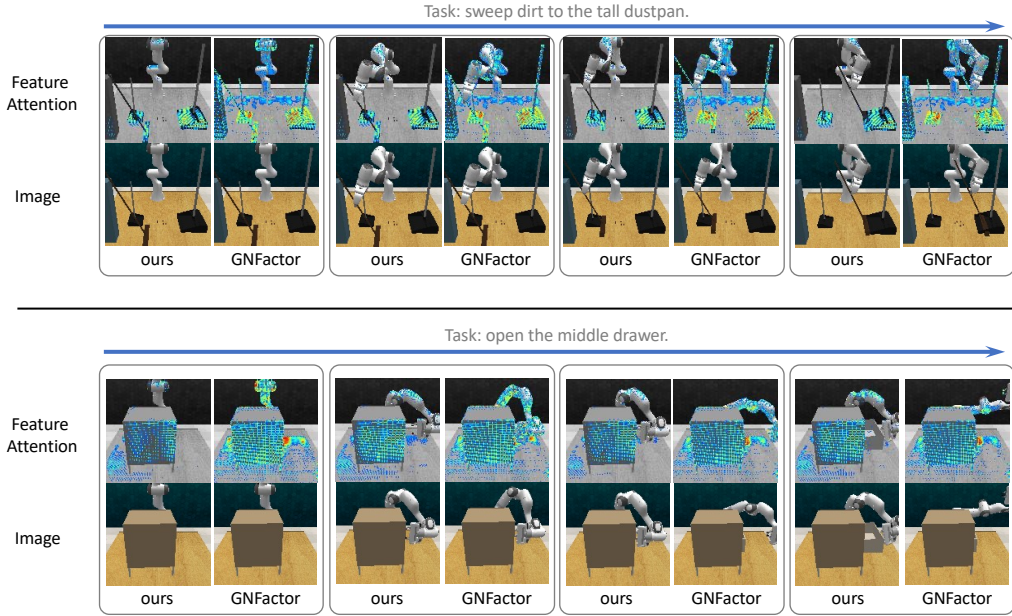


Figure 6: **Visualization Comparison of GNFactor [17] and Ours.**

B.2 Further Experiment Analysis

Experiments on Episode Length. While analyzing the success rate, we also collected statistics on the episode length of the evaluation episodes. The episode length refers to the average number of predicted keyposes or steps in each episode. A shorter episode length indicates fewer steps needed

	Avg. Length	close jar	open drawer	sweep to dustpan	meat off grill	turn tap	slide block	put in drawer	drag stick	push buttons	stack blocks
GNFactor	17.0	19.4	9.9	16.0	15.1	11.0	21.1	16.8	12.8	23.8	23.8
GNFactor w/ Adapter	15.9	19.3	6.7	12.1	14.4	11.7	19.4	20.7	9.1	21.6	23.6

Table 11: **Episode Length.** We report the average episode length across 5 evaluation seeds. As observed, by focusing on the more relevant portion of the scene with the task-aware representations, action efficiency is also improved.

to complete a task, suggesting a more efficient prediction method. As shown in Tab. 11, our method achieves a shorter episode length than GNFactor in most tasks, on average, 6.4% fewer steps across 10 tasks, which implies that the policy network makes more precise predictions and thus performs more efficiently.

Analysis of Reproducing Results of 3D Diffuser Actor. Due to differences in input image resolution, our reproduced results are slightly inferior to those proposed by 3D Diffuser Actor [18]. From the experimental results, it can be observed that for tasks where fine geometric details are crucial, such as "close jar," "meat off grill," and "stack blocks," our reproduced results perform poorly. This is consistent with the lack of detailed information in our data. On the other hand, we find that for tasks that only require determining the general position of an object, such as "put in drawer" and "push buttons", our reproduced results significantly outperform those reported in the original paper. This highlights the significant influence that different representations (determined by input data) have on the current process of robot learning.

Ablation Results Analysis. From the comparison of ablation results, we observe that using only the task objective as the sole condition for the hypernetwork often leads to worse performance than the original codebase. We believe this is because, in RLBench, the ten selected tasks have limited variation in task objectives, with each variation corresponding to a unique task objective. As a result, training tends to lead to the hypernetwork memorizing the task objective rather than generalizing, turning the hypernetwork into a container for memorizing a few sets of parameters instead of a tool for dynamically adjusting the information extraction process. Consequently, the entire network is prone to significant overfitting, leading to poor evaluation results.

Analysis on Limited Improvement Compared to 3D Diffuser Actor Codebase. The experiments show that compared to our significant improvement on GNFactor framework, our method has limited improvement over the 3D Diffuser Actor. Through visualization of the representation compared to the input image, we observe that, compared to the representation of the 3D Diffuser Actor, our representation is primarily focused on task-relevant areas, while the 3D Diffuser Actor’s representation is more dispersed. From this perspective, our representation should significantly outperform that of the 3D Diffuser Actor during task execution. However, the final experimental results show limited improvement. We believe this is because the diffusion policy network has a strong capability for information extraction. During training, the diffusion policy not only extracts task-relevant information from the pre-trained backbone features but also further predicts action outcomes based on this information. Therefore, although our representation is better suited for learning manipulation tasks, the powerful policy network largely bridges the gap. In contrast, when using a relatively less powerful policy network, such as the Perceiver Actor, the performance improvement brought by the HyperTASR becomes much more significant.

Failure Case Analysis. In simulation experiments, the failure usually appears when accurate operation on tiny objects is needed. We conduct experiments on 128×128 resolution and 256×256 resolution, from which we observe that with higher resolution, the average success rate increased from 78.5% to 81.3%. Therefore, we believe that the capability of manipulating tiny objects are highly related to the input sensory data resolution. For real robot experiments, we define task success by finishing the task without significantly changing the position of other unrelated objects in

the scene. In actual evaluation, many failures are caused by changing the position of other unrelated objects due to we do incorporate collision loss in real robot experiments.

B.3 Additional Visualization

We provide additional visualization results to further demonstrate the effectiveness of our approach. First, we compare the gradient visualizations of our method and GNFactor across more tasks in Fig. 6. From these results, it can be observed that, compared to GNFactor, our representation’s attention map is more focused on task-relevant objects, whereas GNFactor’s representation tends to allocate some attention to the background and objects unrelated to the task.

Next, we present a comparison of task execution between our method and GNFactor in Fig. 8. We provide RGB image sequences of the action execution. It can be seen that, compared to GNFactor, our approach more accurately identifies the locations of task-relevant objects, enabling more precise action execution and ultimately leading to successful task completion. In contrast, GNFactor often fails to complete the task due to getting stuck after an incorrect action execution. Meanwhile, we present a comparison of the real-world task execution of 3D Diffuser Actor and our HyperTASR in Fig. 7. More comparison are shown in Supplementary Video.

We also present the change in our representation during the action execution process in GNFactor tasks in Fig. 9. Specifically, for the ”stack blocks” task, our method shows high attention on a target block before placing it, and once the block is successfully placed, the attention on it significantly decreases. This indicates that the information regarding the block becomes less important after its placement in the context of completing the task.

Meanwhile, we provide visualizations of the gradients of our representation versus the input image for the 3D Diffuser Actor in Fig. 10. It is evident that, compared to the 3D Diffuser Actor, our method’s attention is much more concentrated.

In addition, we provide attention visualization of real world experiments in Fig. 11. We compute the gradient of the representation with respect to the input image. We can observe that, compared with the 3D Diffuser Actor with the attention spread through the entire image, HyperTASR is much more focused on task-related objects. Meanwhile, during the task execution, we can observe that initially, attention is focused on the yellow cup and the gripper. As the yellow cup has been picked, the attention switches to the grey cup and the robotic arm. Finally, in the stacking process, the representation focuses on two cups again. This proves our HyperTASR generates representations that dynamically adapt as the task progresses. In Fig. 11, we visualize attention in real robot experiments by computing the gradient of the learned representation with respect to the input image of the training set. Unlike the 3D Diffuser Actor, whose attention is diffusely distributed across the scene, HyperTASR concentrates its attention on task-relevant objects. During the early grasping phase, attention is tightly focused on the yellow cup and the gripper. Once the yellow cup is lifted, attention shifts to the grey cup and the robotic arm. Finally, as the stacking motion commences, the model’s attention returns to both cups. These observations demonstrate that HyperTASR produces dynamic, task-aware representations that track the evolving focus requirements throughout task execution.

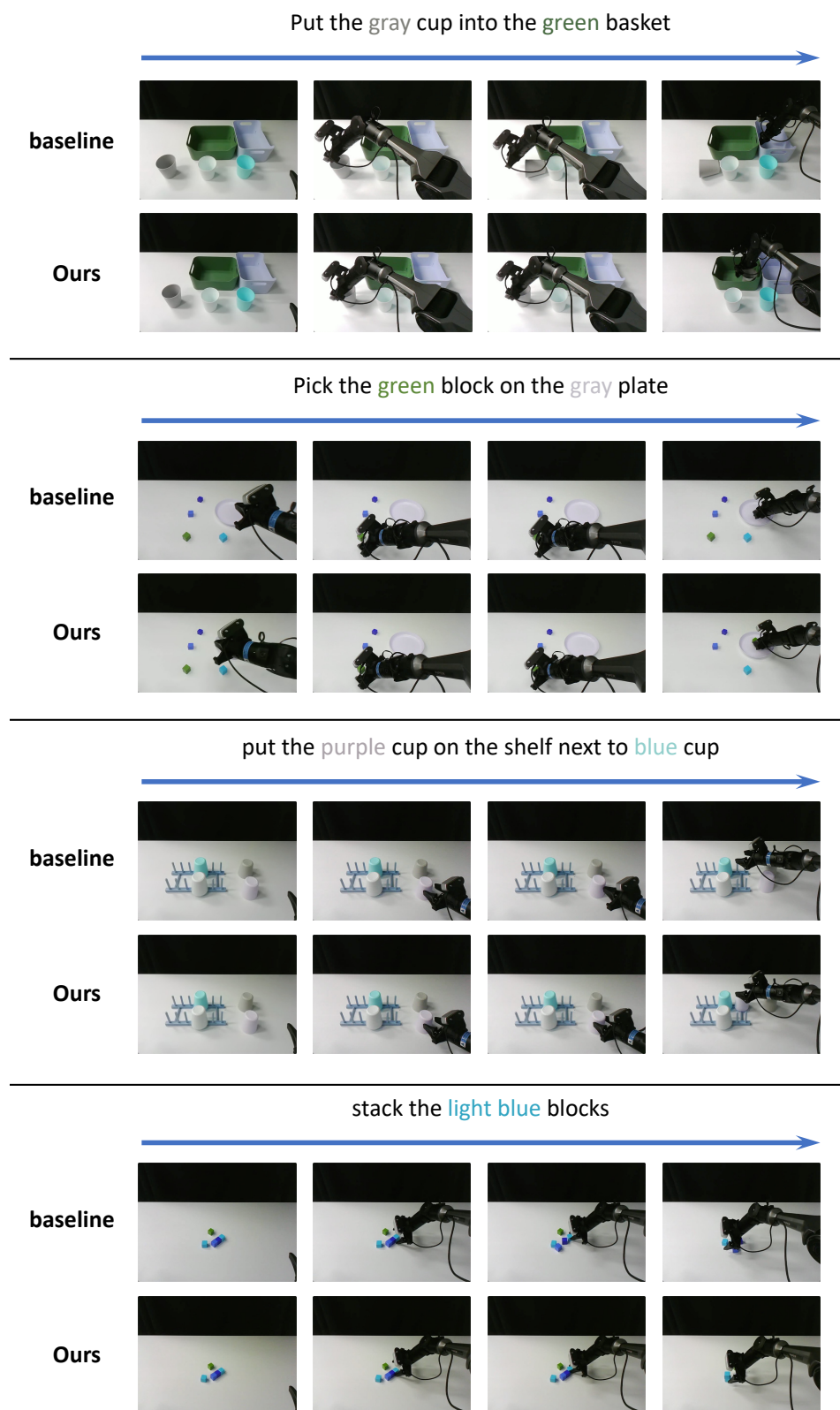


Figure 7: Real World Task Execution Comparison of 3D Diffuser Actor and Ours.

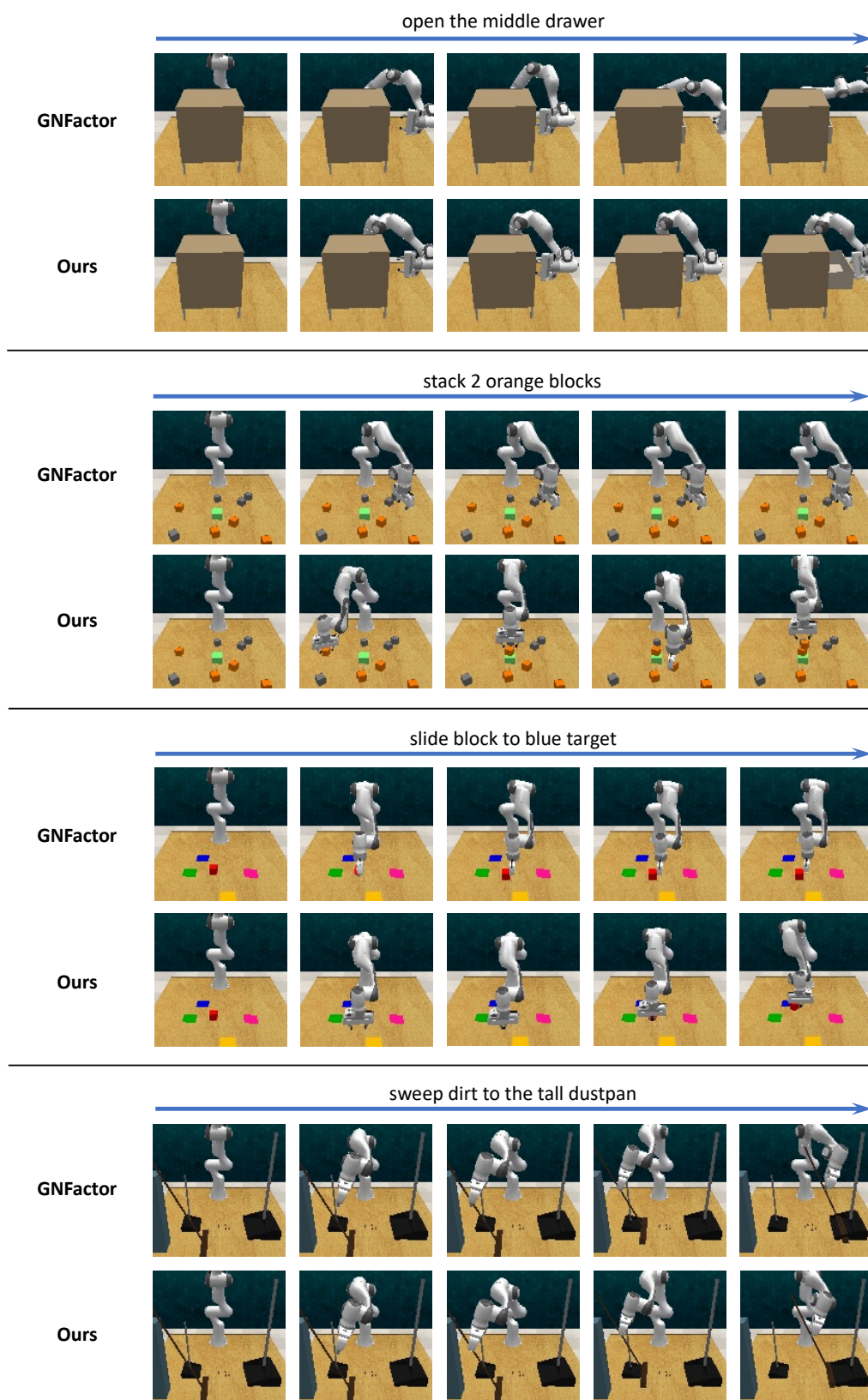


Figure 8: Task Execution Comparison of GNFactor [17] and Ours.

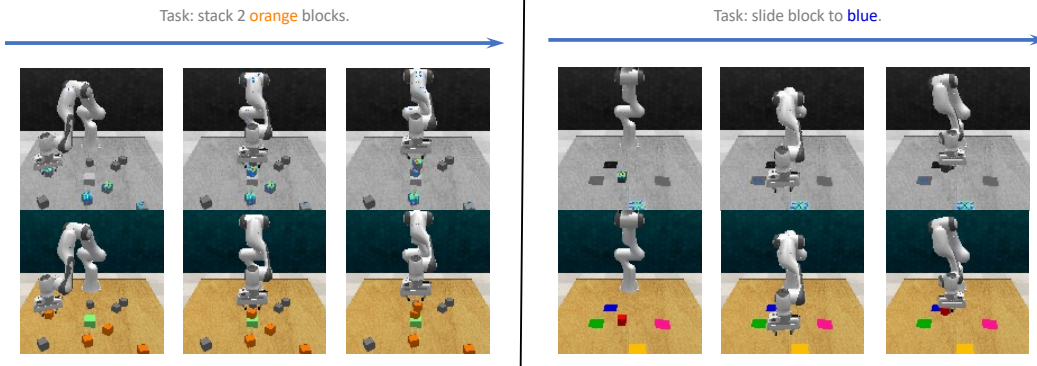


Figure 9: **Visualization Comparison regarding Task Progress for GNFactor [17] with our HyperTASR.**

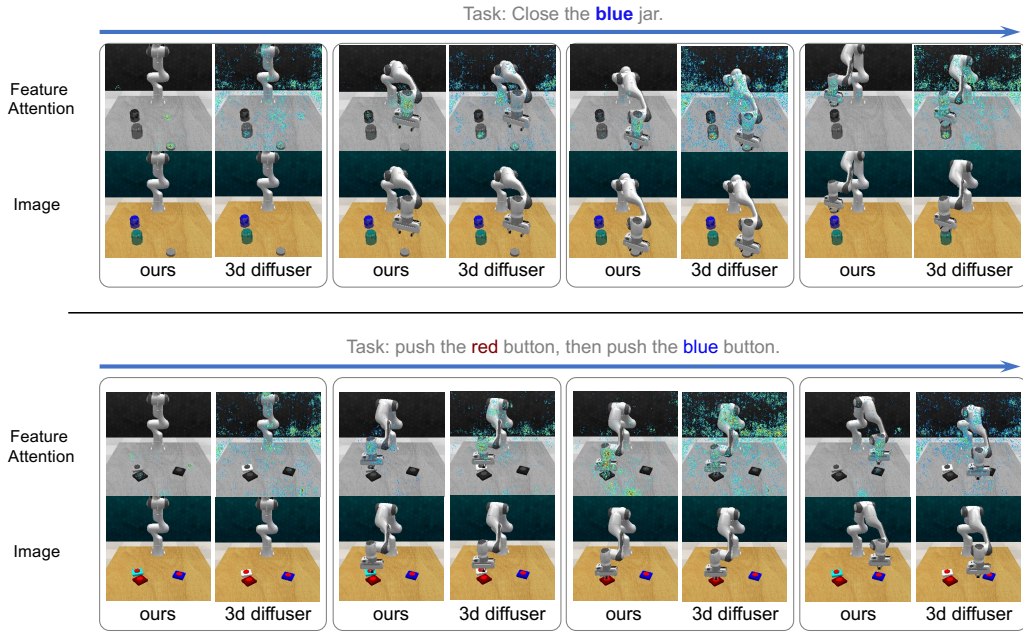


Figure 10: **Visualization Comparison of 3D Diffuser Actor [18] and Ours.**

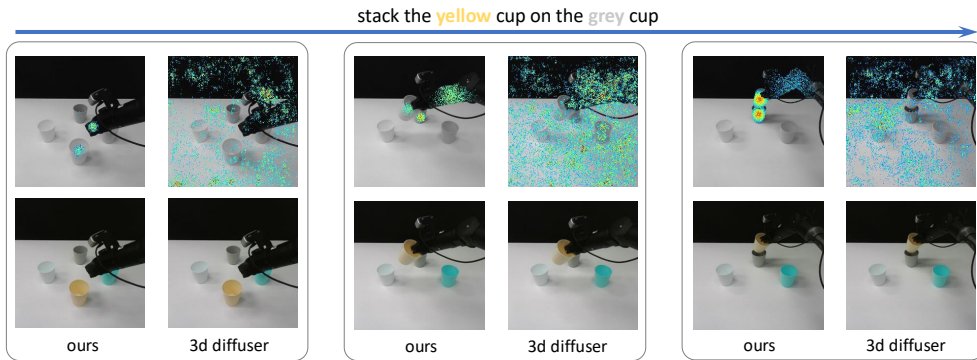


Figure 11: **Real Robot Visualization Comparison of 3D Diffuser Actor [18] and Ours.**