FlowRAM: Grounding Flow Matching Policy with Region-Aware Mamba Framework for Robotic Manipulation

Supplementary Material

In this supplementary material, we provide additional details and experiments not included in the main paper due to space limitations.

A. RLBench Tasks and Training Pipeline

A.1 Details of RLBench Tasks

We present two experimental settings employed in our work. The first involves 10 tasks from RLBench [29], summarized in Tab. A and visualized in Fig. A. Each task encompasses at least two or more variations to evaluate the multi-task capabilities of the agent. Notably, due to the templated nature of the instructions, which vary with each variation, the agent learns a language-guided multi-task policy, rather than learning one-off policies for single variation tasks [59]. Additionally, we evaluate FlowRAM on high-precision tasks by selecting representative tasks from the RLBench benchmark based on [24].

Task	# Variation	# Keyframe	Instruction Template
Close Jar	color(20)	6.0	'close the _ jar'
Open Drawer	placement(3)	3.0	'open the _ drawer'
Sweep to Dustpan	size(2)	4.6	'sweep dirt to dustpan'
Meat off Grill	object(2)	5.0	'take the _ off the grill'
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 stick to drag cube onto the _ target'
Put Buttons	color(50)	3.8	'push the _ button'
Stack Blocks	color, count(60)	14.6	'stack to __ blocks'

Table A. **single-view setting Tasks.** We introduce variations of these tasks, the number of keyframes and instructions template.

A.2 Details of Training Pipeline

To facilitate policy learning, we uniformly sample a set of expert episodes across all task variations. From these sampled episodes, we randomly select input-action pairs for each task to construct a training batch. Similar to previous work [19, 21, 24, 59], the agent is assumed to use a sampling-based motion planner, which helps define input-action pairs as keyframes, as described in Sec. 3.2. *This setup simplifies the original sequential decision-making problem into predicting the next optimal keyframe action based on the current observation.*

B. Details of FlowRAM

The hyperparameters used in FlowRAM are shown in Tab. B. Specifically, $\mathbf{a}_{\text{pos}} \in \mathbb{R}^3$, we use a 6D rotation representation to avoid the inherent discontinuities of quaternions, where $\mathbf{a}_{\text{rot}} \in \mathtt{R}^6$. We apply a Flow Matching Discrete Scheduler for adding noise to both the position $\mathbf{a}_{\text{pos}}$ and the rotation $\mathbf{a}_{\text{rot}}$. To provide a clear understanding of the flow matching policy, we provide pseudocode for flow-based model training Algorithm 1 and inference Algorithm 2.

Hyperparameter	**Value**
Training	
batch size	320
training iteration	300K
learning rate	$1e^{-4}$
weight decay	$5e^{-4}$
optimizer	AdamW
EMA	0.9999
gradient moment	(0.9, 0.999)
loss weight : $\mathcal{L}_{\text{CFM}}$	100
loss weight : $\mathcal{L}_{\text{open}}$	10
Model	
image resolution	128×128
embedding dim C	120
noise scheduler	EulerDiscrete
FlowMatching timestep	50
sampled semantic tokens N_1	4096
sampled geometric tokens N_2	1024
Mamba	
d_model	120
d_state	16
d_conv	4
expand	2

Table B. **Hyper-parameters of FlowRAM.**

C. Additional Experiment and Visualization

C.1 Additional Experiment

To further evaluate the proposed Dynamic Radius Schedule (DRS), we analyze its effect over a few time steps, with qualitative results presented in Tab. C.

i-steps	2	4	8	16	32	
CFM	70.5	73.6	74.0	74.6	76.5	Avg. Sucess
w/o DRS	60.7	94.3	143.1	252.0	499.7	Inference Time
CFM	**74.2**	**77.8**	**78.6**	**79.3**	**81.1**	Avg. Sucess
w DRS	**57.7**	**91.0**	**139.7**	**248.5**	**496.3**	Inference Time

Table C. **Ablation study about the DRS in few timesteps.** Note: If DRS is not used, it means that the global point cloud is downsampled to extract geometric features.

The incorporation of DRS consistently enhances the success rate of the model across all time steps. For example, at $i = 2$, the success rate improves from 70.5% (w/o DRS) to 74.2% (w/ DRS), and at $i = 32$, from 76.5% to

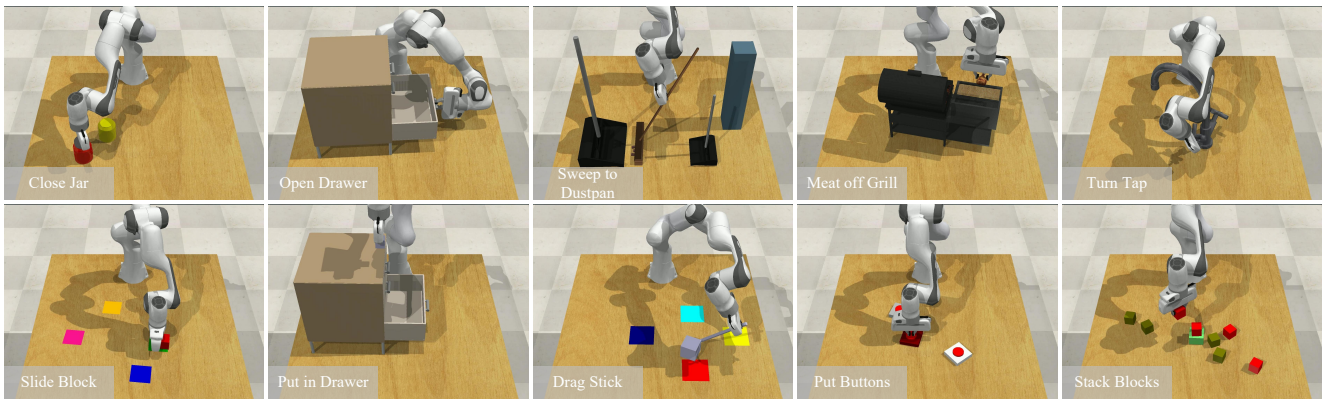

Figure A. **Additional Task Visualization.**We visualize all tasks in the single-view setting.

81.1%, confirming that DRS contributes to higher accuracy and robustness, especially at larger inference steps, while also reducing inference time. The results demonstrate that in robotic manipulation tasks, leveraging local information significantly enhances task success rates. The DRS mechanism dynamically adjusts the radius, enabling the model to focus on task-relevant regions efficiently while minimizing redundant global computations.

Algorithm 1: Conditional Flow Matching Policy Training in Euclidean Space

Require: Dataset $\zeta = \{\boldsymbol{o}, \mathtt{a}\}$, Instructions l, Observations $\boldsymbol{o}$, Conditions $\mathcal{C} = \{\boldsymbol{o}, l\}$, Actions $\mathtt{a} = \{\mathtt{a}_{\text{pos}}, \mathtt{a}_{\text{rot}}\}$

repeat
  $\mathtt{a}_1, \boldsymbol{o} \sim \zeta$ # *Sample a random input-action pair from the dataset*
  $\mathtt{a}_0 \sim \mathcal{N}(0, 1)$ # *Sample from a Gaussian distribution*
  $t \sim \text{Uniform}\{0, \ldots, 1\}$ # *Sample a random time t*
  $\mathtt{a}_t \leftarrow t \cdot \mathtt{a}_1 + (1 - t) \cdot \mathtt{a}_0$ # *Linear interpolation*
  $u(\mathtt{a}_t) \leftarrow \mathtt{a}_1 - \mathtt{a}_0$ # *Compute target velocity Field*
  $\hat{u}(\mathtt{a}_t) \leftarrow \boldsymbol{v}_\theta(\mathtt{a}_t, t, \mathcal{C})$ # *Predicted velocity Field*
  $\mathcal{L}_{\text{CFM}} \leftarrow \|u(\mathtt{a}_t) - \hat{u}(\mathtt{a}_t)\|^2$ # *Compute loss*
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{CFM}}$
until Converged

C.2 Flow Matching Generation process

Fig. B showcases the Flow Matching Generation process of the trained FlowRAM agent in predicting the next keyframe pose, where time step i is set to 50 for a clear depiction of the Flow Matching Generation process. FlowRAM learns accurate velocity fields across various tasks, which facilitates a more efficient flow matching generation process during inference. (For better visualization, we eliminate some of the noise inherent in the depth camera.)

Algorithm 2: Action Poses Generation based on Conditional Flow Matching Policy

Require: Observation $\boldsymbol{o}$, Number of steps k_{steps}, condition $\mathcal{C} = \{\boldsymbol{o}, l\}$

$\mathtt{a}_0 \sim \mathcal{N}(0, 1)$ # Sample from a Gaussian distribution
for $k = 1$ to k_{steps} **do**
  $t \leftarrow k / k_{\text{steps}}$ # Compute time step
  $\Delta t \leftarrow 1 / k_{\text{steps}}$ # Compute interval
  $\boldsymbol{P} \leftarrow \boldsymbol{v}_\theta(z, t, \mathcal{C})$ # Predict velocity
  $\mathtt{a}_t \leftarrow \mathtt{a}_0 + \boldsymbol{P} \cdot \Delta t$ # Update state
end for
return $\mathtt{a}_t$

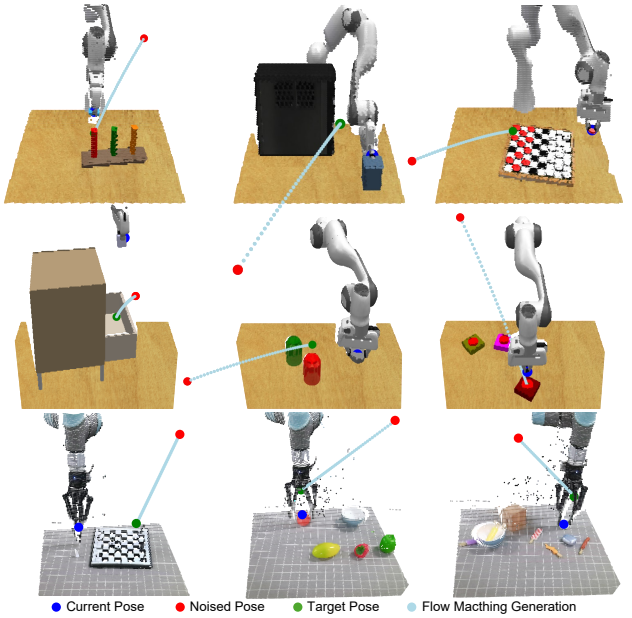

Figure B. **Examples of the next keyframe pose prediction using FlowRAM.** The first two rows represent high-precise and single-view simulation tasks, while the third row corresponds to real-world tasks.

Method	Avg. Success↑	Push Buttons	Slide Block	Sweep to Dustpan	Meat off Grill	Turn Tap	Put in Drawer	Close Jar	Drag Stick	Put in Safe
PerAct	49.4	92.8 ±3.0	74.0 ±13.0	52.0 ±0.0	70.4 ±2.0	88.0 ±4.4	51.2 ±4.7	55.2 ±4.7	89.6 ±4.1	86.0 ±3.2
3D Diffuser Actor	81.3	98.4 ±2.0	97.6 ±3.2	84.0 ±4.4	96.8 ±1.6	99.2 ±1.6	96.0 ±3.6	96.0 ±2.5	100.0 ±0.0	**97.6** ±2.0
RVT-2	81.4	100.0 ±0.0	92.0 ±2.8	**100.0** ±0.0	**99.0** ±1.7	99.0 ±1.7	**96.0** ±0.0	**100.0** ±0.0	99.0 ±1.7	96.0 ±2.8
FlowRAM (ours)	**84.9**	**100.0** ±0.0	**100.0** ±0.0	92.0 ±2.0	94.0 ±2.0	**100.0** ±0.0	92.0 ±0.0	96.0 ±2.0	**100.0** ±0.0	96.0 ±0.0

Method	Place Wine	Screw Bulb	Open Drawer	Stack Blocks	Stack Cups	Put in Cupboard	Insert Peg	Sort Shape	Place Cups	
PerAct	44.8 ±7.8	17.6 ±2.0	88.0 ±5.7	26.4 ±3.9	2.4 ±2.2	28.0 ±4.4	5.6 ±4.1	16.8 ±4.7	2.4 ±3.2	
3D Diffuser Actor	93.6 ±4.8	82.4 ±2.0	89.6 ±4.1	68.3 ±3.3	47.2 ±8.5	85.6 ±4.1	65.6 ±4.1	44.0 ±4.4	24.0 ±7.6	
RVT-2	95.0 ±3.3	**88.0** ±4.9	74.0 ±11.8	**80.0** ±2.8	**69.0** ±5.9	66.0 ±4.5	40.0 ±0.0	35.0 ±7.1	38.0 ±4.5	
FlowRAM (ours)	**96.0**±0.0	84.0±2.3	**92.0** ±0.0	77.3 ±3.8	61.0 ±2.0	**86.0**±4.0	**72.0**±2.7	**48.0**±4.0	**42.0** ±2.3	

Table D. **Evaluation on RLBench with 100 demonstrations.** We report the success rate for 18 RLBench tasks and the average success rate across all the tasks. Additionaly, we show the mean and standard deviation of success rates (in %) average across three random seeds. FlowRAM outperforms all prior arts on most tasks, especially for tasks that require high geometric understanding.

C.3 Multi-Modal Actions

Policies based on generative models allow the modeling of multimodal action distributions [9], i.e., scenarios where there are multiple valid actions given observations and instruction. Fig. C shows a typical example of FlowRAM in multimodal action prediction. With multiple `red blocks` to choose from, there is diversity in the poses of the next keyframe. Therefore, during training, starting from the noise distribution, FlowRAM effectively learns the multimodal distribution of keyframe actions by fitting the vector field along probabilistic flow paths. For inference, one path is chosen for each noise sample and numerical integration is used to generate the end-effector pose for the next keyframe.

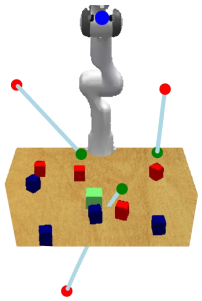

Figure C. **Multi-Modal Predictions.**

D. Experiments on a major benchmark

We understand the importance of the 18-task benchmark in RLBench for evaluating the generalization and robustness of robotic learning algorithms. Therefore, we provide a comprehensive analysis of the results, including success rates, statistical significance, and insights into the performance of FlowRAM. The success rates for each task, along with the average success rate across all tasks, are summarized in Sec. 5. FlowRAM demonstrated excellent performance, consistently achieving high success rates across the majority of tasks.

E. Failure cases and Limitations

E.1 Analysis of Failure cases

In the 17 simulation tasks, `Put umbrella in Stand` task showed the poorest performance. Upon careful examination, we find that while FlowRAM is able to grasp the umbrella handle reliably, two primary errors lead to failure during the insertion process: (1) The umbrella is not correctly inserted into the stand, occurring in approximately 25% of cases; (2) Although the umbrella is successfully inserted into the stand and has detached from the gripper after the two-finger gripper opens, due to a simulator bug, the umbrella continues to follow the center point of the gripper and is pulled out of the stand, occurring in approximately 55% of cases.

In Tab. 1, FlowRAM fails to achieve state-of-the-art performance on the `Sweep to Dustpan` and `Meat off Grill` tasks. Further analysis reveals that at an earlier checkpoint (e.g., 150K training iterations), these tasks achieved success rates of 96% and 92%, respectively, while other tasks remained under-optimized. We hypothesize that the reported results in Tab. 1 may reflect performance trade-offs caused by a uniform task sampling strategy, where equal sampling weights led to performance gains in some tasks at the expense of others [21, 59].

E.2 Limitations

Although FlowRAM has demonstrated remarkable accuracy and scalability in both simulated and real-world scenarios, it still faces challenges in balancing multi-task learning. Future research could focus on exploring advanced strategies for effectively optimizing multi-task learning while maintaining high accuracy.