

# Frame Mining: a Free Lunch for Learning Robotic Manipulation from 3D Point Clouds

## Supplementary Material

### 1 S.1 Supplementary Video and Code

2 Our supplementary video can be viewed at [this link](#), which includes visualizations of learned trajectories for different methods. For FM-MA, we fuse both the robot-base frame and the end-effector frame(s). For each task, an agent is trained on multiple objects.

5 Our code can be viewed at [this github link](#).

### 6 S.2 Architecture of the other two FrameMiners

7 Fig. S1 shows architectures of the other two FrameMiners, FrameMiner-FeatureConcat (FM-FC) and FrameMiner-TransformerGroup (FM-TG).

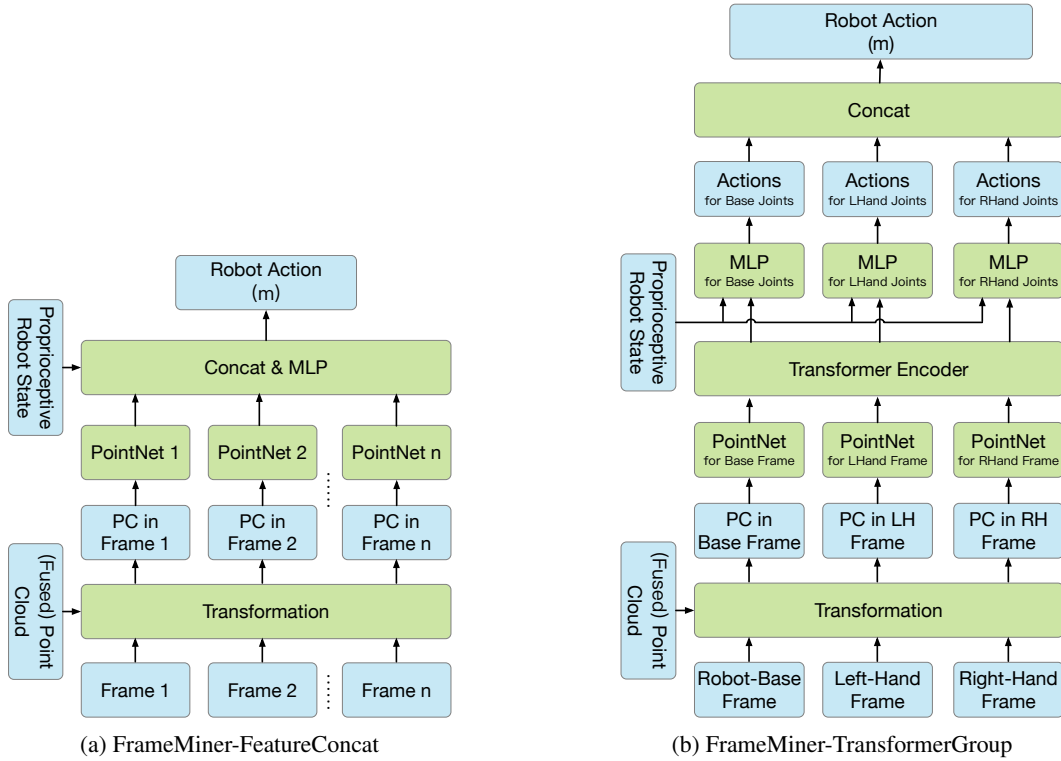


Figure S1: Architectures of FrameMiner-FeatureConcat and FrameMiner-TransformerGroup.

### 9 S.3 More Details of Manipulation Tasks

#### 10 Task Descriptions:

- 11 • In OpenCabinetDoor, a single-arm mobile agent needs to approach a cabinet, use the handle to
- 12 fully open the designated cabinet door, and then keep the door static for a while.
- 13 • In OpenCabinetDrawer, a single-arm mobile agent needs to approach a cabinet, use the handle to
- 14 fully open the designated cabinet drawer, and then keep the drawer static for a while.

- In PushChair, a dual-arm mobile agent needs to approach the chair, push the chair to a target location, and then keep the chair static for a while.
- In MoveBucket, a dual-arm mobile agent needs to approach the bucket, move the bucket to a target platform, place the bucket onto the platform, and then keep the bucket static for a while.
- In PickObject, a single-arm fixed-base agent needs to grasp an object from the table, lift it up to a certain target height, and keep it static for a while.

Simulations are fully physical. For OpenCabinetDoor, OpenCabinetDrawer, PushChair, and MoveBucket, there are 66, 49, 26, and 29 different objects (designated parts) during training, respectively.

### Observations and Actions:

For all ManiSkill tasks, the proprioceptive robot state includes:

- Positions of all (two if single-arm, four if dual-arm) fingers
- Velocities of all (two or four) fingers
- x, y position of the mobile robot base
- Mobile robot base’s rotation around the z-axis
- x, y velocity of the mobile robot base
- Angular velocity of the mobile robot base around the z-axis
- Joint angles of the robot, excluding the joints in the mobile base
- Joint velocities of the robot, excluding the joints in the mobile base
- Indicator of whether each joint receives an external torque

The action space includes:

- x, y velocity of the mobile robot base
- Angular velocity of the mobile robot base around the z-axis
- Height of the robot body
- Joint velocities of the robot, excluding joints of the mobile base and the gripper fingers
- Joint positions of the gripper fingers

Joint positions of the gripper fingers are controlled by position PID. All other action components are controlled by velocity PID.

For the PickObject task, the proprioceptive robot state includes:

- Joint angles of the robot,
- Joint velocities of the robot,
- 1D gripper joint position,
- Target xyz positions of object.

The action space includes 3 DoF end-effector position and 1 DoF gripper joint position.

For all tasks, input point cloud features include xyz coordinates, RGB colors, and one-hot segmentation masks for each part category.

## S.4 Detailed Experimental Settings and Hyperparameters

For our visual backbones, our PointNets are implemented with a three-layer MLP with dimensions [64, 128, 300] followed by a max-pooling layer. We do not apply any spatial transformation to the inputs. Our SparseConvNets are implemented as a SparseResNet10 using TorchSparse [1]. SparseResNet10 has a 4-stage pipeline with kernel size 3 and hidden channels [64, 128, 256, 512] respectively. We use kernel size 3 and stride 2 for downsampling. Initial voxel size is 0.05. Final features in the final-stage voxels are maxpooled as output visual feature.

57 All of our agents are trained with PPO (hyperparameters in Tab. S1). Each policy MLP that outputs  
 58 actions has dimensions  $[192, 128, \text{action\_dim}]$ . For FM-MA that uses input-dependent joint-specific  
 59 weights to fuse action proposals from different frames, the MLP has dimension  $[192, n \times m]$ , where  
 60  $n$  is the number of frames and  $m$  is the dimension of action space. For FM-TG that uses Transformer  
 61 to fuse features from different frames, the Transformer has 3 layers with hidden dimension 300 and  
 62 feed-forward dimension 1024. For all network variants, the value head takes the concatenation of all  
 63 visual features from all frames as input and passes through an MLP with dimensions  $[192, 128, 1]$  to  
 64 output value prediction.

65 For each task, we train an agent for a fixed number of environment steps. Specifically, for OpenCab-  
 66 inetDoor, OpenCabinetDrawer, and MoveBucket, we train for 15 million steps. For PushChair, we  
 67 train for 20 million steps. For PickObject, we train for 4 million steps. Success rates are calculated  
 68 among 300 evaluation trajectories.

Hyperparameters	Value
Optimizer	Adam
Discount ( $\gamma$ )	0.95
$\lambda$ in GAE	0.95
PPO clip range	0.2
Coefficient of the entropy loss term of PPO $_{Cent}$	0.0
Advantage normalization	True
Reward normalization	True
Number of threads for collecting samples	5
Number of samples per PPO update	40000
Number of epochs per PPO update	2
Number of samples per minibatch	330
Gradient norm clipping	0.5
Max KL	0.2
Policy learning rate	3e-4 (non FM-TG); 1e-4 (FM-TG)
Value learning rate	3e-4

Table S1: Hyperparameters for PPO.

## 69 S.5 More Details of Real-World Experiments

70 Fig. S2 shows the captured RGB images and  
 71 point clouds in both simulation and the real  
 72 world (by RealSense camera). For both simula-  
 73 tion and the real-world environment, the ground  
 74 points are removed using z-coordinate thresh-  
 75 old or RANSAC, and the distant points are  
 76 clipped. To reduce the sim-to-real gap, we only  
 77 use xyz coordinates as our input point cloud  
 78 feature, and we discard RGB colors.

79 We also demonstrate our real-world experi-  
 80 ments at the end of our supplementary video.

## 81 References

- 82 [1] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin,  
 83 H. Wang, and S. Han. Searching effi-  
 84 cient 3d architectures with sparse point-  
 85 voxel convolution. In *European Confer-*  
 86 *ence on Computer Vision*, 2020.

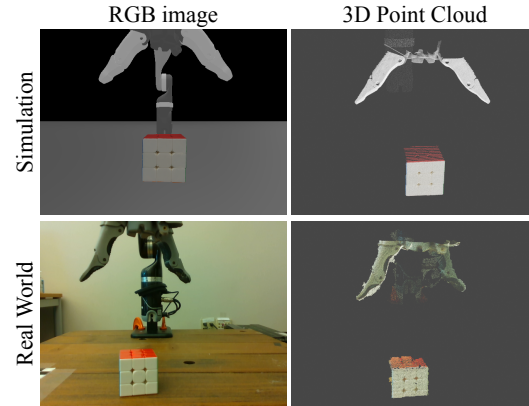


Figure S2: RGB images and 3D point clouds captured in both simulation and the real world. Colored point clouds for better illustration.