A. Environment settings

A.1. Meta-world

We compare our approach with other baselines in Meta-world [10] for addressing complex robot manipulation tasks. An agent observes only visual images from the environment, without using sensor data. The agent decides the velocities in the x, y, and z axes of the grippers and the gripping force for action to perform the specified manipulation task. We utilize the reward function defined in Meta-world [10]. We considered the difficulty of tasks and selected 10 tasks (button-press, door-lock, door-unlock, drawer-close, faucet-open, faucet-close, reach, reach-wall, window-close, handle-press) from the MT50 benchmark for our experiments. Figure A.1 illustrates states in the Meta-world environment.

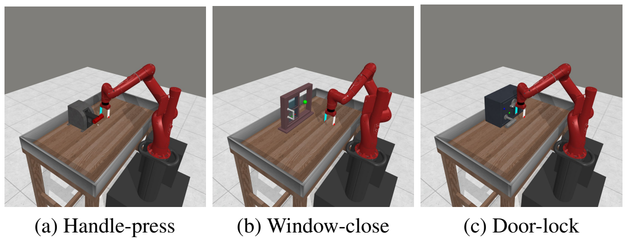
(a) Handle-press (b) Window-close (c) Door-lock

Figure A.1. Visualization examples of tasks in Meta-world

A.2. CARLA

To verify the performance of models at mission-critical autonomous driving tasks with inference time constraint, we conduct experiments in the CARLA simulator [3]. CARLA is an open-source simulator for autonomous driving tasks, providing various environment settings for driving maps. The agent observes visual image, and sensor data such as acceleration, angular velocity, location, rotation, direction, velocity, target location, and previous action. The agent is capable of performing actions such as accelerating, braking, and steering.

For the agent's position $pos(t)$ at timesteps t and $col(t)$ is the collision penalty, the reward is calculated by

$$R(s, a, T) = ||pos(t) - T||_2 - col(t) \tag{A.1}$$

where T is specified by the target location. We use 12 different maps as a multi-task. Figure A.2 shows examples of driving maps. Considering the complexity of autonomous driving, we supplement the learning algorithm with online behavior cloning algorithms in CARLA.

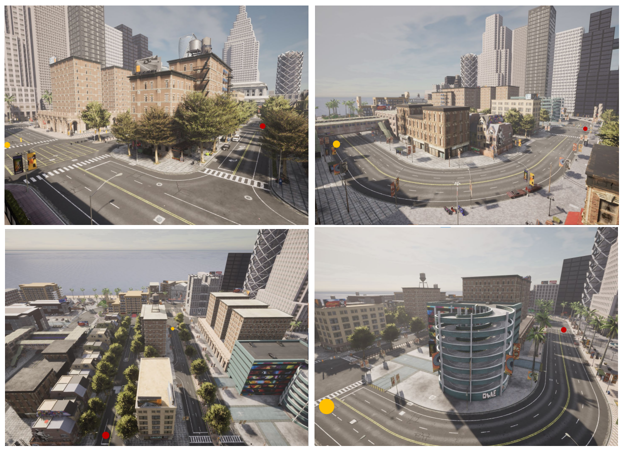
Figure A.2. Visualization examples of maps in CARLA, Yellow dots denote start locations and Red dots denote target locations.

A.3. AI2THOR

To evaluate the applicability of our model in complex navigation tasks in an embodied environment and a time constraint, we perform experiments in AI2THOR [4]. We use rearrange tasks [8] in AI2THOR simulator, which places objects to match a given goal image. The agent receives both the current visual image and the position of target objects, which is indicated in the goal image, as task information. And, the agent performs high-level semantic actions in this environment demonstrated in Table A.1. We train our model with the reward functions defined in [8]. For the pre-trained model, we utilize the expert dataset provided from [8] with DAgger algorithms [7]. We define the tasks for multi-task evaluation based on the positions of the target objects and conduct evaluations on 50 seen and 50 unseen scenarios. Example images of AI2THOR are illustrated in Figure A.3.

Table A.1. AI2THOR Actions

Navigation	Move [Ahead/Left/Right/Back] Rotate [Right/Left] Look [Up/Down] Done
Object interaction	PickUp [Object Type] Open [Object Type] PlaceObject

Figure A.3. Visualization examples in AI2THOR

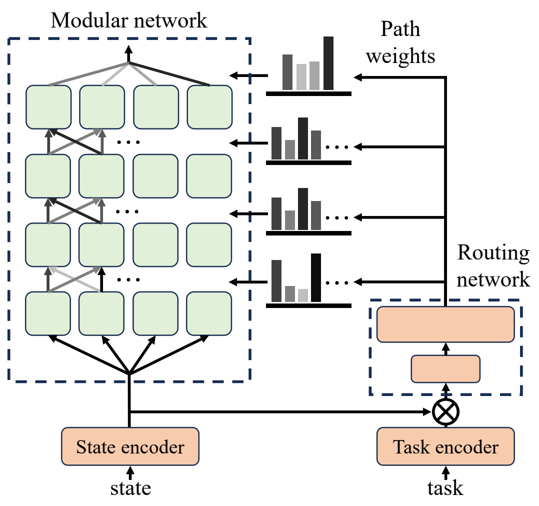

Figure A.4. Base network architecture of MoDeC

B. Implementations

In this section, we provide the implementation details of our model and comparison methods. Our mode is implemented using Python v3.8, Jax v0.3.24, and Pytorch v1.13. It is trained on a system of an Intel(R) Core(TM) i9-10940X CPU @ 3.30GHz and NVIDIA GeForce RTX 4090. And device adaptor is trained and evaluation is performed on three embedded devices, Nvidia Jetson Nano (Nano), Nvidia Jetson Xavier NX 8GB (Xavier), and Nvidia Jetson AGX Orin 32GB (Orin).

B.1. MoDeC (Ours)

The base network consists of the modular network and soft routing network. In the base network, to be the same as [9], the modular network takes the output of the state encoder, and the routing network gets the output of the state and task encoders as input. Overall architecture is shown as Figure A.4 In our implementation, the actor network only uses the modular network architecture. It is because that the critic is only used for training, and then the critic does not need a dynamic inference. The hyperparameter settings for the network architecture of the base network π_{base} are detailed in Table A.2.

Iterative module selection network and module selection use the same network architecture. Also, the actor and critic share a feature extractor, and each of them has two fully connected (fc) layers at the end. Joint learning is implemented as the base network and the iterative module selection network is trained alternately for a certain number of steps. The network architecture of the iterative module selection network and module selection network is in Table A.3, and the hyperparameter settings for training are detailed in Table A.4.

Table A.2. The network architecture of the base network π_{base} for MoDeC.

Configurations			**Values**
Num. of layer			4
Num. of module per layer			4
Activation function			Swish
Actor	State encoder		[6×6, 2, 32] (for CARLA) [6×6, 2, 64] (for otherwise)
	Task encoder		64-d fc (for CARLA) 128-d fc (for otherwise)
	Routing network		maxpool 2×2 [6×6, 2, 16]
			[6×6, 2, 64] (for CARLA) [6×6, 2, 128] (for otherwise)
			64-d fc×2 (for CARLA) 128-d fc×2 (for otherwise)
			52-d fc
	Modular network	layer 1	maxpool 2×2
			[3×3, 1, 32] (for CARLA) [3×3, 1, 128] (for otherwise)
		layer 2	maxpool 2×2
			[3×3, 1, 32] (for CARLA) [3×3, 1, 128] (for otherwise)
		layer 3	maxpool 2×2
			[3×3, 1, 64] (for CARLA) [3×3, 1, 256] (for otherwise)
		layer 4	maxpool 2×2
			[3×3, 1, 64] (for CARLA) [3×3, 1, 256] (for otherwise)
	fc layers		64-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)
Critic network			maxpool 4×4
			[6×6, 2, 32] (for CARLA) [6×6, 2, 64] (for otherwise)
			maxpool 2×2
			[6×6, 2, 64] (for CARLA) [6×6, 2, 128] (for otherwise)
			avgpool 5×5
			64-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)

Table A.3. Network architectures of iterative module selection network π_{ims} and module selection network π_{ms} for MoDeC

Configurations	**Values**
Activation function	Swish
Feature extractor	maxpool 4×4
	[6×6, 2, 32] (for CARLA) [6×6, 2, 64] (for otherwise)
	maxpool 2×2
	[6×6, 2, 64] (for CARLA) [6×6, 2, 128] (for otherwise)
	avgpool 5×5
Actor network	64-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)
Critic network	64-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)

B.2. DRNet

DRNet [1] is a model consisting of cells formed by acyclic graphs between nodes connected in a series. In this model, the overall architecture is a sequence of multiple cell struc-

Table A.4. Training hyperparameters of MoDeC

Configurations	Values
Batch size	300 (for iter. module sel. net.) 60 (for otherwise)
Temperature τ	1.0 (for iter. module sel. net.) 0.005 (for otherwise)
Timesteps	3e+7 (for base network) 1e+7 (for iter. module sel. net.) 1e+6 (for module sel. net.)
Train frequency	300 (for iter. module sel. net.) 12 (for otherwise)
Gradient steps per update	1
Discount factor γ	0.99
Buffer size	1e+5 (for base network)
Learning rate	1e-4
Device adapter	
Num. of data	1e+3
Batch size	16
Epochs	100
Learning rate	1e-3

tures, which consists of input nodes, intermediate nodes, output nodes, branches, and RouterNet. Every input node is connected by a set of branches to intermediate nodes, and the output node is calculated by the summation of intermediate nodes. RouterNet dynamically determines the relative importance weight of each branch among the branches of each connection. In our implementation, we use the CNN layer and skip-connection layer as the branches. The actor and critic share DRNet parameters, and each of them has two fully connected (fc) layers at the end. In training DRNet, we control the trade-off between inference time and performance by the coefficient of CNN usage penalty based on the environment reward. Network architecture for DRNet is detailed in the Table A.5, and the training hyperparameter settings are detailed in the Table A.6.

Table A.5. Network architecture of DRNet.

Configurations		Values
Num. of cell		2
Num. of input node		1
Num. of intermediate node		4
Activation function		Swish
Cell 1	Branch	[3×3, 1, 32] (for CARLA) [3×3, 1, 128] (for otherwise)
	RouterNet	[3×3, 1, 32] (for CARLA) [3×3, 1, 128] (for otherwise)
		avgpool, 16-d fc
Cell 2	Branch	[3×3, 1, 64] (for CARLA) [3×3, 1, 256] (for otherwise)
	RouterNet	[3×3, 1, 64] (for CARLA) [3×3, 1, 256] (for otherwise)
		avgpool, 16-d fc
Actor network		64-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)
Critic network		64-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)

Table A.6. Training hyperparameters of DRNet.

Configurations	Values
Learning rate	1e-4
Temperature τ	0.005
Discount factor γ	0.99
Train frequency steps	12
Gradient steps per update	1
Batch size	60
Buffer size	1e+5
Timesteps	3e+7

B.3. D2NN

D2NN [6] is a dynamic network architecture composed of regular and control nodes, where the control node dynamically determines which regular nodes that actual operations are performed to use for inference. Similar to DRNet, this model trains the best performance-efficiency trade-off through reward shaping by CNN usage penalty. We use the same network architecture of MoDeC for D2NN. That is, the regular node corresponds to each module in the base network, and the control node has the same network architecture as the module selection network. The hyperparameter settings are detailed in the Table A.7.

Table A.7. Training hyperparameters of D2NN.

Configurations	Values
Learning rate	1e-4
Batch size	60 (for regular node) 300 (for control node)
Temperature	0.005 (for regular node) 1.0 (for control node)
Train frequency	12 (for regular node) 300 (for control node)
Timesteps	3e+7 (for regular node) 1e+7 (for control node)
Discount factor γ	0.99
Gradient steps per update	1
Buffer size	1e+5 (for regular node)

B.4. DS-Net

DS-Net [5] is a dynamic slimable model which slices its own CNN network channels by ratio, the output of a gater network. We slightly modify this to suit our problem situation, replacing the ratio with an input instead of using the gater network. In a similar context, as the dynamic architecture of DS-Net is unnecessary for the critic, this model is included only in the actor. The network architecture and hyperparameter settings are detailed in the Table A.8, A.9.

B.5. RL-AA

RL-AA [2] is a cost-aware RL model where the high-level policy dynamically determines which policy to use from the low-level policy, a set of policies with various sizes of network architectures. For a wide range of real-time inference

Table A.8. Network architecture of DS-Net.

Configurations			**Values**
Activation function			Swish
Actor	State encoder		[6×6, 2, 32] (for CARLA) [6 × 6, 2, 256] (for otherwise)
	DS-Net	Layer 1	[3×3, 1, 32]×2 (for CARLA) [3×3, 1, 256]×2 (for otherwise)
			batchnorm, maxpool 2×2
		Layer 2	[3×3, 1, 64]×2 (for CARLA) [3×3, 1, 512]×2 (for otherwise)
			batchnorm, maxpool 2×2
	fc layers		avgpool 5×5
			64-d fc (for CARLA) 512-d fc (for otherwise)
			32-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)
Critic network			maxpool 4×4
			[6×6, 2, 32] (for CARLA) [6×6, 2, 64] (for otherwise)
			maxpool 2×2
			[6×6, 2, 64] (for CARLA) [6×6, 2, 128] (for otherwise)
			avgpool 5×5
			32-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)

Table A.9. Hyperparameters of DS-Net

Configurations	**Values**
Available ratio	[0.125, 0.25, 0.5, 0.75, 1.0]
Activation function	ReLU
Learning rate	1e-4
Temperature τ	0.005
Discount factor γ	0.99
Train frequency steps	12
Gradient steps per update	1
Batch size	60
Buffer size	1e+5
Timesteps	3e+7

time adjustments, we expand the number of low-level policies and modify the high-level policy to select a low-level policy at every step. A set of low-level policies is generated by utilizing the network architectures consisting of up to i^{th} layer of the biggest low-level policy network. The size of the network output features varies depending on the number of low-level policies, thus the filter size of the last average pooling layer is adjusted accordingly. Detailed network architecture is described in the Table A.10. The hyperparameter settings are detailed in the Table A.11.

Table A.10. Network architecture of RL-AA

Configurations		**Values**
Activation function		Swish
High-level policy	Layer 1	maxpool 4×4
		[6×6, 2, 32] (for CARLA) [6×6, 2, 64] (for otherwise)
	Layer 2	maxpool 2×2
		[6×6, 2, 64] (for CARLA) [6×6, 2, 128] (for otherwise)
		avgpool 5×5
	Actor network	64-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)
	Critic network	64-d fc×2 (for CARLA) 256-d fc×2 (for otherwise)
Low-level policy	Layer 1	[3×3, 1, 32] (for CARLA) [6×6, 2, 128] (for otherwise)
		maxpool 2×2
	Layer 2	[3×3, 1, 64] (for CARLA) [6×6, 2, 256] (for otherwise)
		maxpool 2×2
	Layer 3	[3×3, 1, 96] (for CARLA) [6×6, 2, 384] (for otherwise)
		maxpool 2×2
	Layer 4	[3×3, 1, 128] (for CARLA) [6×6, 2, 512] (for otherwise)
		maxpool 2×2
	avgpool	
	Actor network	32-d fc×2 (for CARLA) 128-d fc×2 (for otherwise)
	Critic network	32-d fc×2 (for CARLA) 128-d fc×2 (for otherwise)

Table A.11. Hyperparameters of RL-AA

Configurations	**Values**
Number of layer for low-level policy	[1, 2, 3, 4]
Efficiency coefficient	3e-5
Learning rate	1e-4
Temperature τ	1
Discount factor γ	0.99
Train frequency steps	1
Gradient steps per update	1
Batch size	60
Buffer size	1e+5
Timesteps	3e+7

C. Additional results

C.1. Inference time

All inference times for MoDeC used in the experiments is the end-to-end value, which includes not only the base network but also the module selection network and device adapter. To clarify the impact of adjusting the inference time of the base network on the model, Table A.12 provides a detailed analysis of inference time for experiments conducted on the Orin device.

C.2. Environment with varying time constraint

Table A.13 shows the performance with varying time constraints in CARLA. In the environment, the time constraint

Table A.12. Percentage of inference time in Nvidia Jetson AGX Orin

Constraint	Base network	Module selection net.	Device adapter
8 ms	4.91 ms / 62.6%	2.43 ms / 30.9%	0.50 ms / 6.4%
10 ms	6.93 ms / 70.1%	2.45 ms / 24.9%	0.50 ms / 5.0%
12 ms	8.84 ms / 74.9%	2.47 ms / 20.9%	0.50 ms / 4.2%
14 ms	10.80 ms / 78.4%	2.50 ms / 18.0%	0.50 ms / 3.6%
16 ms	11.64 ms / 79.5%	2.50 ms / 17.1%	0.50 ms / 3.4%

is sampled from a normal distribution with a specific mean and standard deviation. We conduct experiments on two devices with various mean and standard deviation settings. As shown, our approach outperforms DS-net by 2.2% to 11.6%. Since MoDeC dynamically determines the module combination based on task information and module utilization, it achieves higher performance through more efficient and flexible module selection compared to the ratio-based slicing in DS-Net, showing a greater performance difference $5.4\% \sim 11.6\%$ under intense time constraints.

Table A.13. Performance on CARLA with varying time constraints

Device	Constraint (ms)		DS-Net	MoDeC
	Mean	Std.	Success rate	Success rate
Orin	10	2	$45.8 \pm 6.3\%$	$51.2 \pm 7.2\%$
	12	1	$75.0 \pm 4.0\%$	$77.2 \pm 5.1\%$
	12	2	$69.3 \pm 8.1\%$	$75.1 \pm 5.2\%$
	12	4	$67.1 \pm 7.2\%$	$71.9 \pm 7.0\%$
	14	2	$75.1 \pm 5.2\%$	$79.3 \pm 6.7\%$
Xavier	15	4	$37.9 \pm 9.1\%$	$49.5 \pm 7.9\%$
	18	1.5	$73.9 \pm 4.3\%$	$81.4 \pm 4.2\%$
	18	4	$68.4 \pm 8.3\%$	$74.5 \pm 6.1\%$
	18	6	$69.0 \pm 6.5\%$	$72.3 \pm 7.1\%$
	21	4	$72.7 \pm 5.8\%$	$77.3 \pm 6.1\%$

References

[1] Shaofeng Cai, Yao Shu, and Wei Wang. Dynamic routing networks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3588–3597, 2021. 2

[2] Chin-Jui Chang, Yu-Wei Chu, Chao-Hsien Ting, Hao-Kang Liu, Zhang-Wei Hong, and Chun-Yi Lee. Reducing the deployment-time inference control costs of deep reinforcement learning agents via an asymmetric architecture. In *Proceedings of the 38th IEEE International Conference on Robotics and Automation (ICRA)*, pages 4762–4768. IEEE, 2021. 3

[3] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Proceedings of the 1st Conference on Robot Learning (CoRL)*, pages 1–16. PMLR, 2017. 1

[4] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017. 1

[5] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *Proceedings of the 23nd IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8607–8617, 2021. 3

[6] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Proceedings of the 34nd AAAI Conference on Artificial Intelligence*, 2018. 3

[7] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011. 1

[8] Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual room rearrangement. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 1

[9] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. In *Proceedings of the 34th conference on neural information processing systems (NeurIPS)*, pages 4767–4777, 2020. 2

[10] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proceedings of the 4th conference on robot learning (CoRL)*, pages 1094–1100. PMLR, 2020. 1