Appendix

A Implementation Details

In this section, we describe the implementation details of our algorithm for training on the training view and test time training in view generalization settings on the DMControl [30], xArm [7], and Adroit [20] environments. We utilize the official implementation of TD-MPC [11] and MoDem [8] which are available at github.com/nicklashansen/tdmpc and github.com/facebookresearch/modem as the model-based reinforcement learning codebase. During training time, we use the default hyperparameters in official implementation of TD-MPC and MoDem. We present relevant hyperparameters during both training and test time in Table 10 and Table 11. One seed of our experiments could be run on a single 3090 GPU with fewer than 2GB and it takes $\sim$ 1 hours for test-time training.

Training time setup. We train visual model-based policies with TD-MPC on DMControl and xArm environments, and MoDem on Adroit environments, We employ identical network architecture and hyperparameters as original TD-MPC and MoDem during training time.

The network architecture of the encoder in original TD-MPC is composed of a stack of 4 convolutional layers, each with 32 filters, no padding, stride of 2, 7×7 kernels for the first one, 5×5 kernels for the second one and 3×3 kernels for all others, yielding a final feature map of dimension $3 \times 3 \times 32$ (inputs whose framestack is 3 have dimension $84 \times 84 \times 9$). After the convolutional layers, a fully connected layer with an input size of 288 performs a linear transformation on the input and generates a 50-dimensional vector as the final output.

The network architecture of the encoder in original Modem is composed of a stack of 6 convolutional layers, each with 32 filters, no padding, stride of 2, 7×7 kernels for the first one, 5×5 kernels for the second one and 3×3 kernels for all others, yielding a final feature map of dimension $2 \times 2 \times 32$ (inputs whose framestack is 2 have dimension $224 \times 224 \times 6$). After the convolutional layers, a fully connected layer with an input size of 128 performs a linear transformation on the input and generates a 50-dimensional vector as the final output.

Test time training setup. During test time, we train spatial adaptive encoder (SAE) to adapt to view changes. We insert STN blocks before and after the first convolutional layer of the original encoders in TD-MPC and MoDem. The original encoders are augmented by inserting STN blocks, resulting in the formation of SAE. Particularly, for the STN block inserted before the first convolutional layer, the input is a single frame. This means that when the frame stack size is N, N individual frames are fed into this STN block. This is done to apply different transformations to different frames in cases of moving and shaking view.

To update the SAE, we collect online data using a buffer with a size of 256. For each update, we randomly sample 32 (observation, action, next_observation) tuples from the buffer as a batch. The optimization objective is to minimize the loss in predicting the dynamics of the latent states, as defined in Equation 2.

During testing on each task, we run 20 consecutive episodes, although typically only a few or even less than one episode is needed for the test-time training to converge. To make efficient use of the data collected with minimal interactions, we employ a multi-update strategy. After each interaction with the environment, the SAE is updated 32 times.

The following is the network architecture of the first STN block inserted into the encoder of TD-MPC.

```
STN_Block_0_TDMPC(
  (localization): Sequential(
    # By default, each image consists of three channels. Each frame in the
        ↪ observation is treated as an independent input to the STN.
    (0): Conv2d(in_channels=3, out_channels=8, kernel_size=7, stride=1)
    (1): MaxPool2d(kernel_size=4, stride=4, padding=0)
    (2): ReLU()
    (3): Conv2d(in_channels=8, out_channels=10, kernel_size=5, stride=1)
```

```
      (4): MaxPool2d(kernel_size=4, stride=4, padding=0)
      (5): ReLU()
    )
    (fc_loc): Sequential(
      (0): Linear(in_dim=90, out_dim=32)
      (1): ReLU()
      (2): Linear(in_dim=32, out_dim=6)
    )
  )
```

The following is the network architecture of the second STN block inserted into the encoder of TD-MPC.

```
  STN_Block_1_TDMPC(
    (localization): Sequential(
      (0): Conv2d(in_channels=32, out_channels=8, kernel_size=7, stride=1)
      (1): MaxPool2d(kernel_size=3, stride=3, padding=0)
      (2): ReLU()
      (3): Conv2d(in_channels=8, out_channels=10, kernel_size=5, stride=1)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0)
      (5): ReLU()
    )
    (fc_loc): Sequential(
      (0): Linear(in_dim=90, out_dim=32)
      (1): ReLU()
      (2): Linear(in_dim=32, out_dim=6)
    )
  )
```

The following is the network architecture of the first STN block inserted into the encoder of MoDem.

```
  STN_Block_0_MoDem(
    (localization): Sequential(
      # By default, each image consists of three channels. Each frame in the
          ↪ observation is treated as an independent input to the STN.
      (0): Conv2d(in_channels=3, out_channels=5, kernel_size=7, stride=2)
      (1): MaxPool2d(kernel_size=4, stride=4, padding=0)
      (2): ReLU()
      (3): Conv2d(in_channels=5, out_channels=10, kernel_size=5, stride=2)
      (4): MaxPool2d(kernel_size=4, stride=4, padding=0)
      (5): ReLU()
    )
    (fc_loc): Sequential(
      (0): Linear(in_dim=90, out_dim=32)
      (1): ReLU()
      (2): Linear(in_dim=32, out_dim=6)
    )
  )
```

The following is the network architecture of the second STN block inserted into the encoder of MoDem.

```
  STN_Block_1_MoDem(
    (localization): Sequential(
      (0): Conv2d(in_channels=32, out_channels=8, kernel_size=7, stride=2)
      (1): MaxPool2d(kernel_size=3, stride=3, padding=0)
      (2): ReLU()
      (3): Conv2d(in_channels=8, out_channels=10, kernel_size=5, stride=2)
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0)
      (5): ReLU()
    )
    (fc_loc): Sequential(
      (0): Linear(in_dim=90, out_dim=32)
      (1): ReLU()
      (2): Linear(in_dim=32, out_dim=6)
    )
  )
```

Table 10: **Hyperparameters for training time.**

Hyperparameter	Value
Discount factor	0.99
Image size	84×84 (TD-MPC) 224×224 (MoDem)
Frame stack	3 (TD-MPC) 2 (MoDem)
Action repeat	1 (xArm) 2 (Adroit, Finger, and Walker in DMControl) 4 (otherwise)
Data augmentation	± 4 pixel image shifts (TD-MPC) ± 10 pixel image shifts (MoDem)
Seed steps	5000
Replay buffer size	Unlimited
Sampling technique	PER ($\alpha = 0.6$, $\beta = 0.4$)
Planning horizon	5
Latent dimension	50
Learning rate	1e-3 (TD-MPC) 3e-4 (MoDem)
Optimizer (θ)	Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$)
Batch size	256
Number of demos	5 (MoDem only)

Table 11: **Hyperparameters for test time training.**

Hyperparameter	Value
Buffer size	256
Batch size	32
Multi-update times	32
Learning rate for encoder	1e-6 (xArm) 1e-7 (otherwise)
Learning rate for STN blocks	1e-5

B Environment Details

We categorize the view generalization problem into four distinct settings: novel view, moving view, shaking view, and novel FOV. In this section, we provide descriptions of the implementation details for each setting. The detailed camera settings can be referred to in the code of the environments that we are committed to releasing or in the visualization available on our website movie-rl.github.io.

Novel view. In this setting, for locomotion tasks (cheetah-run, walker-stand, walker-walk, and walker-run), the camera always faces the moving agent, while for other tasks, the camera always faces a fixed point in the environment. Therefore, as we change the camera position, the camera orientation also changes accordingly.

Moving view. Similar to the previous setting, the camera also always faces the moving agent or a fixed point in the environment. The camera position varies continuously.

Shaking view. To simulate camera shake, we applied Gaussian noise to the camera position (XYZ coordinates in meters) at each time step. For DMControl and Adroit, the mean of the distribution is 0, the standard deviation is 0.04, and we constrain the noise within the range of -0.07 to +0.07. For xArm, the mean of the distribution is 0, the standard deviation is 0.4, and we constrain the noise within the range of -0.07 to +0.07.

Novel FOV. We experiment with a larger FOV. For DMControl, we modify the FOV from 45 to 53. For xArm, we modify the FOV from 50 to 60. For Adroit, we modify the FOV from 45 to 50. We also experiment with a smaller FOV and results are presented in Appendix E.

C Visualization of Feature Map Transformation

We visualize the first layer feature map of the image encoder from TD-MPC and MoVie in Figure 7. It is observed that the feature map from MoVie on the novel view exhibits a closer resemblance to that on the training view.

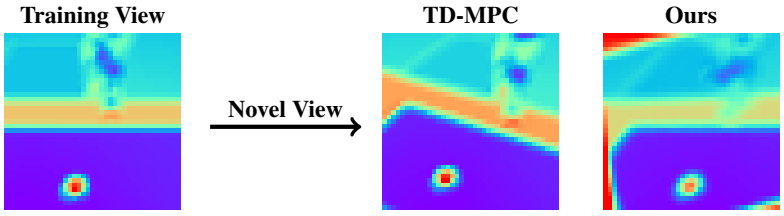

Figure 7: **Visualization of the first layer feature maps of the original encoder on the training view and on the novel view, and the learned SAE on the novel view.**

D Extended Description of Baselines

TD-MPC. We test the agent trained on training view without any adaptation in the view generalization settings,

DM. This is derived from MoVie by removing STN blocks, which just adapts encoder during test time.

IDM+STN. This is derived from MoVie by replacing the dynamics model with the inverse dynamics model which predicts the action in between based on the latent states before and after transition. The inverse dynamics model is finetuned together with the encoder and STN blocks during testing.

E Ablation on Different FOVs

In our main experiments, we consider the novel FOV as a FOV larger than the original. In Table 12, we present results for both smaller and larger FOV scenarios. Our method demonstrates the successful handling of both cases.

Table 12: **Ablation on different FOVs.** The best method on each setting is in **bold**.

Cheetah-run	TD-MPC	DM	IDM+STN	MoVie
Small FOV	104.85±4.59	398.75±17.52	75.92±8.76	**530.37±12.84**
Large FOV	128.55±6.57	379.01±10.90	299.02±88.47	**532.94±19.74**

F Results of Original Models on Training View

The performance of the original agents without any adaptation under the training view is reported in Table 13, 14, and 15 for reference. In the context of view generalization, it is evident that the performance of agents without adaptation significantly deteriorates.

Table 13: **Training Result on DMControl.**

Task	Cheetah, run	Walker, walk	Walker, stand	Walker, run	Cup, catch	Finger, spin
Reward	$658.10_{\pm 9.98}$	$944.99_{\pm 21.71}$	$983.53_{\pm 5.34}$	$697.75_{\pm 11.35}$	$980.56_{\pm 4.33}$	$985.20_{\pm 2.25}$

Task	Finger, turn-easy	Finger, turn-hard	Pendulum, swingup	Reacher, easy	Reacher, hard
Reward	$756.16_{\pm 150.76}$	$616.96_{\pm 149.44}$	$827.26_{\pm 61.72}$	$983.8_{\pm 0.34}$	$937.43_{\pm 54.59}$

Table 14: **Training Result on xArm.**

Task	Reach	Push	Peg in box	Hammer
Success rate (%)	$96_{\pm 5}$	$90_{\pm 17}$	$80_{\pm 10}$	$83_{\pm 20}$

Table 15: **Training Result on Adroit.**

Task	Door	Hammer	Pen
Success rate (%)	$96_{\pm 3}$	$78_{\pm 7}$	$48_{\pm 15}$