A Additional Implementation Details

We describe all the details of our model implementation aside from the ones mentioned in main text.

A.1 Model Details

Region and Context Features Computation. For encoding images, we use ResNet-18 [46] as the backbone. To obtain a spatial feature map with sufficient resolution for feature pooling, we remove the last four layer of ResNet-18, it produces a spatial feature map with the size of 16×16 and we perform ROI Align [17] to get the pooled features of size 6×6. In order to map the pooled features into the same size as input tokens, we add a linear layer to project the pooled feature. We already mentioned in the main text that we obtain *global context featurex* by passing the spatial feature maps through a Spatial Softmax layer. For eye-in-hand images, we use a Resnet-18 backbone followed by Spatial Softmax to get *eye-in-hand features*. We use one linear layer to map from robots' states to *proprioceptive features*.

Sensor Modalities All the models including VIOLA use the same set of sensor modalities, which are the RGB images from the workspace camera, RGB images from the eye-in-hand camera, joint configuration, and binary gripper state. We use the eye-in-hand camera following the same setup as in the previous work on imitation learning for manipulation [27]. This camera view has been empirically shown to contribute significantly to the performance of visuomotor policies [27, 59, 60].

Neural Network Details. We use a standard Transformer [22] architecture in our paper. We use 4 layers of transformer encoder layers, and 6 heads of the multi-head self-attention modules. For the two-layered fully connected networks, we use 1024 hidden units for each layer. For GMM output head, we choose the number of modes for gaussian mixture model to be 5, which is the same as in Mandlekar et. al. [27].

Positional Encoding. We provide details about both positional features for regions and the temporal positional encoding. For all the encodings, they have the same dimensionality D as the input tokens. For a region's positional feature, we compute it based on its bounding box positions $bbox_pos = (x_0, y_0, x_1, y_1)$:

$$
\begin{aligned}
PE(bbox_pos, 4i) &= f(\frac{x_0}{10^{4i/D}}) \\
PE(bbox_pos, 4i+1) &= f(\frac{y_0}{10^{(4i+1)/D}}) \\
PE(bbox_pos, 4i+2) &= f(\frac{x_1}{10^{(4i+2)/D}}) \\
PE(bbox_pos, 4i+3) &= f(\frac{x_2}{10^{(4i+3)/D}})
\end{aligned}
$$

and f is the sine function when i is even and cosine function when i is odd.

For computing temporal positional encoding, we follow the equation for each dimension i in the encoding vector at a temporal position pos:

$$
\begin{aligned}
PE(pos, 2i) &= \sin(\frac{pos}{10^{2i/D}}) \\
PE(pos, 2i+1) &= \cos(\frac{pos}{10^{(2i+1)/D}})
\end{aligned}
$$

We choose the frequency of positional encoding to be 10 that is different from the one in the original transformer paper. This because our input sequence is much shorter than those in natural language tasks, hence we choose a smaller value to have sufficiently distinguishable positional features for input tokens.

Data Augmentation We applied both color jittering and pixel shifting to VIOLA and all the baselines. As for random erasing, it is used for VIOLA and its variants. We applied random erasing to VIOLA to prevent the policy from overfitting to region proposals. We did not apply this augmentation for baseline behavioral cloning models such as BC-RNN as it yields lower performance than without using random erasing.

For random erasing, we use the open-sourced Torchvision function whose parameters are: $p = 0.5$, scale $= (0.02, 0.05)$, ratio $= (0.5, 1.5)$, value $=$ random.

For applying color jittering to the data, we apply color jittering with brightness=0.3, contrast=0.3, saturation=0.3, hue=0.05 on 90% of the trajectories. We keep 10% unchanged so that we still keep the main visual cue patterns from demonstrations. We do 4 pixel shifting as in Mandlekar et al. [27].

Aside from data augmentation on images, we also add very small gaussian noise on the proposal positions to add more variety on proposal data.

Training Details. In all our experiments of VIOLA, we train for 50 epochs. We use a batch size of 16 and a learning rate of 10^{-4}. We use negative log likelihood as the loss function for action supervision loss since we use a GMM output head. As we notice that validation loss doesn't correlate with policy performance [27], we use a pragmatic way of saving model checkpoint, which is to save the checkpoint that has the lowest loss over all the demonstration data at the end of training. As we use e Transformer as a model backbone, we apply several optimization technique that is suitable for training transformer. We use AdamW optimizer [] along with cosine annealing scheduler of learning []. We also apply gradient clip [22]: 10 for all tasks but 0.1 for the two long-horizon tasks, namely `BUDS-Kitchen` and `Make-Coffee`.

For training baselines, we take the general configs from Mandlekar et al., which is 500 epochs for BC and 600 epochs for BC-RNN, and follow the same model saving criteria as ours. The benchmark study of approaches in Mandlekar et al. [27] report the highest performance of all checkpoints across training, as the loss values do not correlate with policy performance. We recognize that this criteria can inflate the model performance, and it's not pragmatic to evaluate all model checkpoints during real robot experiments. Therefore, we adopt the saving criteria that is same as Zhu et al. [4], where the policies that have lowest training loss on demonstration datasets are taken. As we show in our experiments, our training procedure of VIOLA can easily find a good-performance checkpoint consistently across simulation and real-world.

B Additional Experiments

Here we compile two additional experiments to support some of our design.

B.1 Choice of K in simulation

While simulation images have a huge domain gap from real world, we find that RPN can still localize objects on simulated images given its good priors of "objectness". Here we show the preliminary experiment to decide to choose $K = 20$ in simulation experiments. We quantitatively compute the coverage of object proposals from RPN over objects in the scene (including robots) and we evaluate the coverage with recall rates of object proposals. The recall rates in simulation is easy to compute as ground-truth object bounding boxes are easily accessible in simulation. We iterate K from 5 to 40 with an interval of 5. We compute recall rates of proposals using IoU$= 0.5$, meaning that we consider an object covered if IoU of a proposal bounding box and the ground-truth bounding box is larger than 0.5. We can see that when $K = 20$, we have recall rates that is larger than 70%, which is considered large coverage. Moreover, we can see that when K is larger than 20, the marginal increase of recall rates is only 1% every 5 more proposals. So we choose $K = 20$ in our simulation experiments.

K	5	10	15	20	25	30	35	40
`Sorting`	59.0	69.0	72.0	**74.0**	75.0	76.0	77.0	77.0
`Stacking`	67.0	74.0	77.0	**79.0**	80.0	81.0	82.0	83.0
`BUDS-Kitchen`	72.0	81.0	83.0	**84.0**	85.0	86.0	86.0	87.0

Table 3: Recall rates (%) of object proposals from pre-trained RPN on all simulation environments with IoU$= 0.5$.

B.2 Visual Feature Design

In our work, we learn a spatial feature map from scratch for extracting visual features, aiming to learn actionable visual features that are informative for continuous control. To show the importance of learning actionable visual features from scratch and that pre-trained visual features in visual tasks

are not sufficient for control tasks, we conduct an ablative experiment by comparing our model with two variants that use the pre-trained feature map from Feature Pyramid Network (FPN) in RPN without fine-tuning, and fine-tuning. The result is presented in Table 4. The table suggests that directly using pre-trained spatial feature map without fine-tuning are overall worse in performance than learning from scratch. And fine-tuning the feature map on the downstream tasks doesn't give a matching performance as learning from scratch. This shows the importance of optimizing actionable visual-features for manipulation tasks. Fine-tuning FPN gives an increase in performance to almost match our original model, but doesn't outperform it. This shows that pre-trained visual features are not critical to the model performance, not to mention that the trainable parameters in FPN takes about 200 MB while ours convolutional encoder for spatial feature maps only take 14 MB trainable parameters.

Models	`Canonical`	`Placement`	`Distractor`	`Camera-Jitter`
VIOLA(From Scratch)	87.6 ± 1.1	68.3 ± 1.5	74.4 ± 5.7	50.7 ± 0.6
w/o Fine-tuned FPN	79.7 ± 1.5	52.7 ± 1.2	61.8 ± 3.3	53.7 ± 1.8
w/ Fine-tuned FPN	84.9 ± 1.1	60.4 ± 78.4	76.7 ± 5.1	46.7 ± 1.7

Table 4: Comparison among models that use spatial feature maps learned from scratch, pre-trained spatial feature masks without fine-tuning, and with fine tuning.

C Environment Details

Here we describe the details of environments, including how the testing variants are defined.

Overall setup We use a 7-DoF Franka Emika Panda arm in all tasks, and we use Operational Space Controller [61] for end-effector control, a binary command control for parallel-gripper 20Hz. We use Kinect Azure as the workspace camera and Intel Realsense D435i as the eye-in-hand camera.

Task Details For the `Sorting` task, the robot needs to pick up two boxes successively and place them together in the sorting bin. For the `Stacking` task, the robot needs to stack the same type of boxes in the target region. For the `BUDS-Kitchen` task, the robot needs to place the bread in the pot and serve it after putting on the stove, and turn off the stove after serving. In `Dining-PlateFork`, the robot needs to push the plate into the target region and place the fork next to the plate. The `Dining-Bowl` task requires the robot to push the turntable around and pick up the bowl and place it on the plate. In `Make-Coffee`, the robot is tasked with an entire coffee-making procedure from opening up the k-cup holder to pushing the button to activate the espresso machine.

Data Collection We use a 3Dconnexion SpaceMouse to collect 100 human-teleoperated demonstrations for each simulation task and 50 for each real-world task. We apply color augmentation [62, 63] to the visual observations in demonstrations to increase the visual diversity in datasets, making learned policy robust to visual cue variations due to surrounding lighting conditions.

Task Success Determination The success rates of models are computed based on the success of all evaluated rollouts. In the simulation, the success is determined by whether the object states in the simulator meets the pre-programmed goal function. For example, in **Sorting** task, the goal function is programmed to check if the two boxes are both in the bin, and the simulation environment decides a rollout is successful only if the goal function returns true value. In the real world, the success of a rollout is determined by whether the objects are in the goal configuration, which is implicitly specified in the given demonstrations. We have added this detail in Appendix C (see Ln 588 - Ln 592) to clarify this point.

Evaluation Horizons Based on the collected demonstrations, we determine the evaluation horizons of our simulation tasks in the case of `Canonical`, which are: 1) 1000 for `Sorting`, 800 for `Stacking`, and 1500 for `BUDS-Kitchen`. And for evaluating in testing variants, we have 200 steps more than in `Canonical` for each task.