CS 188 Final Project Report

Incremental Policy Improvement for Robot Cabinet Opening

Andy Kasbarian | UCLA CS 188 | Spring 2026

1. Problem Definition

This project is based on the OpenCabinet starter project by Holden Grissett. The goal is to train a robot policy that controls a PandaOmron mobile manipulator to open cabinet doors in the RoboCasa simulation environment.

The PandaOmron robot is a Franka Panda 7-DOF arm mounted on an Omron wheeled base. It must open a HingeCabinet fixture door from a closed position. The task is considered successful when the door joint reaches at least 90% of its full open range. For this project, success was relaxed to require only one door to be open, rather than all doors, as recommended by the course instructor.

The task is difficult for several compounding reasons:

- Scene diversity: over 2,500 kitchen layouts and styles are used during evaluation. A policy that memorizes a single approach will fail to generalize.
- Sparse reward: the reward signal is binary (0 or 1). There is no partial credit for getting close to the handle or partially opening the door.
- State representation limitations: the low-dimensional state vector (joint positions, end-effector pose, base pose) does not contain visual information about the scene. The robot cannot see which kitchen it is in.
- Precise manipulation required: the robot must locate the handle, grasp it, and apply force in the correct direction, all from a low-dimensional state representation.

Given these constraints, the goal of this project was redefined from simply achieving a high success rate to understanding how much improvement each architectural change contributes when working within the fundamental limitation of state-only input.

2. Method

All policies were trained on the official OpenCabinet human demonstration dataset in LeRobot format, consisting of 107 episodes of expert teleoperation across diverse kitchen scenes. The dataset was augmented with additional handle features before training any policy beyond the baseline.

2.1 Dataset Augmentation (05b)

The baseline dataset contains only robot proprioceptive state: gripper joint positions, base position and orientation, and end-effector position and orientation. Critically, it contains no

information about where the cabinet handle is. Script 05b replays the saved MuJoCo states from each demonstration episode to extract:

- handle_pos (3D): world position of the target door handle
- handle_to_eef_pos (3D): vector from end-effector to handle
- door_openness (1D): normalized door joint angle, 0 = closed, 1 = fully open
- handle_xaxis (3D): the handle body's local x-axis, encoding door facing direction
- hinge_direction (1D): +1 for right-opening doors, -1 for left-opening

This extends the state vector from 16 dimensions to 27 dimensions. All subsequent policies use this augmented state.

2.2 Baseline MLP (06)

The baseline policy is a 3-layer MLP with hidden dimension 256 and Tanh output activation. It takes a single state vector as input and predicts a single action vector. Training uses MSE loss between predicted and demonstrated actions, Adam optimizer, and constant learning rate over 50 epochs.

2.3 MLP with Temporal Context (06a)

The first improvement stacks the last k=4 state observations as input, giving the network a short history window: [s_{t-3}, s_{t-2}, s_{t-1}, s_t] → a_t. This allows the network to implicitly infer velocity and direction of motion from the difference between consecutive states.

Additional improvements over the baseline include: LayerNorm after the first projection for training stability, a residual connection in the middle block, AdamW optimizer with weight decay, and a cosine annealing learning rate schedule. At the start of each episode, the history buffer is zero-padded rather than padded with copies of the first state, which avoids a distribution mismatch at episode start.

A bug was identified post-training in the original 06a implementation: state columns were not being extracted in a consistent order between training and inference, causing the state vector to be scrambled at eval time. This was fixed in all subsequent policies by locking the column order at training time and saving it in the checkpoint.

2.4 Action Chunking (06b)

Instead of predicting one action per timestep, 06b predicts the next K=8 actions simultaneously: [s_{t-3}, ..., s_t] → [a_t, a_{t+1}, ..., a_{t+7}]. These actions are executed open-loop before the policy is queried again. This is the key idea behind ACT (Zhao et al., 2023) and directly addresses the temporally incoherent behavior of single-step prediction.

The model architecture is identical to 06a except the output head is widened to produce K * action_dim values. At inference, a FIFO queue holds the predicted chunk and actions are dequeued one per timestep until the queue is empty, at which point the policy is re-queried.

2.5 MLP Diffusion Policy (06c)

Rather than directly regressing actions with MSE loss, 06c uses denoising diffusion probabilistic modeling (DDPM). The training objective changes from predicting the correct action to predicting the Gaussian noise that was added to a ground-truth action chunk at a randomly sampled diffusion timestep t.

The model is a small MLP that takes as input the concatenation of: the current state vector, the noisy action chunk (flattened), and a sinusoidal embedding of the diffusion timestep t. At inference, the policy starts from pure Gaussian noise and iteratively denoises for 100 steps using the DDPM reverse process to produce a clean action chunk.

The motivation for diffusion is that MSE loss averages multi-modal demonstrations (e.g., approaching the handle from the left vs. the right) into a useless mean action. Diffusion learns a distribution over actions and samples from it, preserving both modes.

2.6 U-Net Diffusion Policy (06d)

The MLP noise predictor in 06c is replaced with a 1D convolutional U-Net (ConditionalUnet1D, adapted from Chi et al., 2023). The action chunk is now treated as a sequence of shape (chunk_size, action_dim) rather than a flat vector, allowing the convolutional layers to exploit temporal structure between consecutive actions.

The U-Net consists of an encoder that progressively downsamples the action sequence (down_dims = [128, 256, 512]), a bottleneck, and a decoder that upsamples back to the original shape. Skip connections bridge each encoder stage to its corresponding decoder stage, preserving fine-grained action details that would otherwise be lost in the bottleneck.

State conditioning is implemented via FiLM (Feature-wise Linear Modulation). At every residual block throughout the network, a small linear layer projects the concatenated state and timestep embedding into a per-channel bias that is added to the intermediate features. This injects contextual information at every layer rather than only at the input.

The U-Net has approximately 10.5 million parameters and was trained for 300 epochs on all 107 episodes, taking approximately 3 hours on an NVIDIA GeForce RTX 3060 Mobile GPU.

3. Evaluation Results

All policies were evaluated over 20 episodes on the pretrain kitchen split using a maximum of 500 steps per episode. Success was defined as any single cabinet door reaching at least 90% of its full open range, as recommended by the course instructor to avoid penalizing policies on multi-door cabinet variants.

Policy	Architecture	Success Rate	Avg Episode Length	Qualitative Behavior
Baseline	MLP, single-step	0%	500 steps	Random flailing
06a	MLP + temporal context	0%	500 steps	Backwards motion (bug)
06b	MLP + temporal + chunking	0%	500 steps	Purposeful motion

06c	MLP diffusion	0%	500 steps	Random flailing
06d	U-Net diffusion	0%	500 steps	Purposeful motion

All policies achieved a 0% success rate, consistent with the expected performance of state-based policies on this benchmark. The course instructor noted in the project README that the MLP baseline "will basically always fail" and that production-quality results require image-based methods such as Diffusion Policy, pi-0, or GR00T N1.5, which achieve 30-60% success rates on the same task.

Despite identical quantitative results, there is a clear qualitative divide. The baseline MLP, 06a (due to the state scrambling bug), and 06c exhibit random flailing behavior with no apparent directionality. By contrast, 06b and 06d both exhibit purposeful initial motion toward the cabinet handle. This is most clearly visible in the 06d visualization, where the robot arm moves toward the handle location before breaking down.

4. Discussion and Reflections

The fundamental bottleneck in this project is the absence of visual input. The state vector, even with the 11-dimensional handle augmentation from 05b, does not capture enough information about scene geometry to generalize across 2,500+ kitchen layouts. The robot cannot see which direction the door opens, where the handle is relative to obstacles, or how the kitchen is oriented. Image-based policies address this directly by conditioning on camera observations.

The most impactful single improvement was action chunking (06b). Both policies that showed purposeful behavior — 06b and 06d — use action chunking, while single-step policies (baseline, 06a, 06c) do not. This suggests that temporal coherence is a more critical property than model architecture or training objective for this task. Action chunking commits the robot to a motion trajectory rather than recomputing a new action every timestep, which prevents the jittery, self-canceling behavior of single-step policies.

The addition of diffusion in 06c did not improve on the MLP baseline, which is expected given that both use single-step prediction. The theoretical advantage of diffusion — handling multi-modal demonstrations — cannot manifest without sufficient temporal coherence. When diffusion was combined with action chunking in 06d, the results were comparable to 06b, suggesting that at this scale and with state-only input, the U-Net architecture does not provide meaningful gains over the simpler MLP chunking approach.

A consistent rotational behavior was observed in the 06d visualization: the robot arm twists clockwise (from the camera perspective) before losing track of the handle. A possible explanation is a sign convention inconsistency in the handle_to_eef_pos feature. If the vector pointing from the end-effector toward the handle is interpreted inversely by the policy at inference time, the robot would consistently move away from the handle in a systematic rather than random direction. This hypothesis was not verified due to time constraints.

If this project were repeated, the priority change would be to incorporate visual observations from the start. The RoboCasa benchmark provides three camera feeds per episode (left shoulder, right shoulder, wrist) at 256x256 resolution. Even a simple CNN encoder on the wrist camera would likely improve performance dramatically by giving the policy direct visual feedback on handle proximity and door state.

5. Team Contributions

This project was completed as a solo effort. The originally assigned partner withdrew from the project last minute.

Contributions by Andy Kasbarian:

- Ran the full project pipeline from environment verification (00) through dataset download and playback (04-05)
- Ran script 05b_augment_handle_data.py (provided by the instructor) to extract handle features from demonstration episodes
- Implemented and trained all four improved policies (06a, 06b, 06c, 06d), including debugging checkpoint key mismatches and inference loop errors
- Implemented evaluation and visualization scripts for all policies (07a-07d, 08a-08d), including adapting the success criteria to check per-door joint state
- Analyzed and documented the qualitative behavioral differences between policies through video visualization
- Built the project website and wrote this report

6. AI Disclosure

Claude (Anthropic) was used extensively throughout this project to write and debug the training scripts (06a-06d), evaluation scripts (07a-07d), and visualization scripts (08a-08d). Claude also explained the conceptual foundations of each method — temporal context, action chunking, DDPM diffusion, and the U-Net architecture with FiLM conditioning — which the author then verified and understood before writing this report.

Script 05b_augment_handle_data.py was provided by the course TA Holden Grissett and was not written by the author or AI.

All code was reviewed, run, and debugged by the author. The analysis, conclusions, and report writing are the author's own work informed by the results observed during experimentation.

References

Chi, C., Feng, S., Du, Y., Xu, Z., Cousineau, E., Burchfiel, B., & Song, S. (2023). Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. RSS 2023. https://diffusion-policy.cs.columbia.edu/

Zhao, T. Z., Kumar, V., Levine, S., & Finn, C. (2023). Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. arXiv:2304.13705.

Nasiriany, S., Maddukuri, A., Zhang, L., Parikh, A., Lo, A., Jain, A., Abbeel, P., & Zhu, Y. (2024). RoboCasa: Large-Scale Simulation of Everyday Tasks for Generalist Robots. https://robocasa.ai/